

Java开发利器

珍藏版



附：视频讲解
本书源代码

强锋科技

那 静 编著

Eclipse SWT/JFace

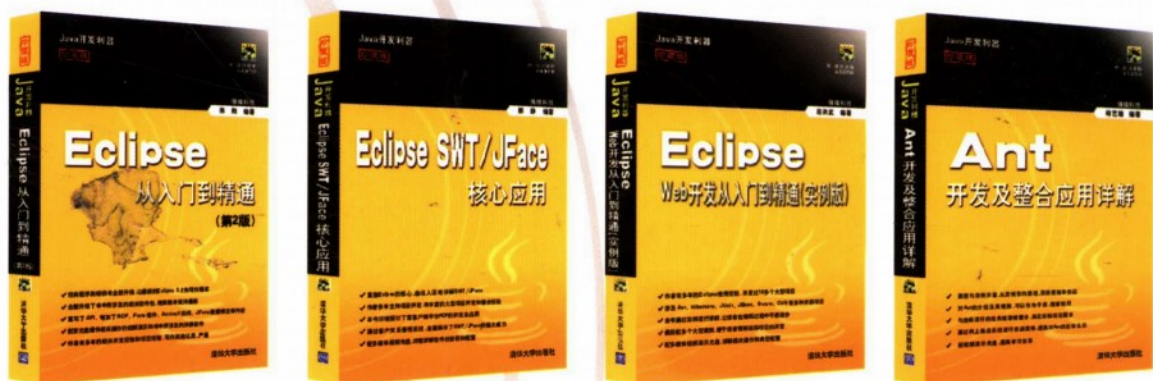
核心应用

- ✓ 直指 Eclipse 的核心，由浅入深地讲解 SWT/JFace
- ✓ 作者多年主持项目开发，有丰富的大型项目开发和整合经验
- ✓ 本书详细探讨了富客户端平台 RCP 的开发及应用
- ✓ 通过客户关系管理系统，全面展示了 SWT/JFace 的强大威力
- ✓ 配多媒体视频光盘，详细讲解软件的安装和配置



清华大学出版社

Eclipse SWT/JFace 核心应用



丛书特色

- ✓ 作者全都是有丰富编程经验的一线开发人员
- ✓ 全面攻克 Java 开发领域的前沿技术
- ✓ 极大地提升您的 Java 应用开发水准
- ✓ 确保技术的实用性和深入性
- ✓ 贯穿丰富的应用实例，真正做到学以致用
- ✓ 配视频演示光盘讲述界面操作，既节省篇幅，又容易上手

光盘内容

- ✓ 全程录制本书涉及软件的基本操作
- ✓ 书中所涉及的程序源代码

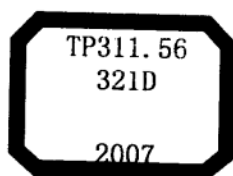
ISBN 978-7-302-14717-6



9 787302 147176 >

定价: 69.00 元(附光盘 1 张)

Java 开发利器



Eclipse SWT/JFace 核心应用

强锋科技

那 静 编著

清华大学出版社

北 京



内 容 简 介

本书全面介绍了 SWT、JFace 和 RCP 的相关知识。全书共分 5 篇,第 1 篇介绍了 SWT 产生的背景以及 SWT 的一些基本概念和基础知识。第 2 篇介绍了 SWT 基本控件的使用,以及事件处理、布局等 SWT 基本知识的应用。第 3 篇介绍了关于 SWT 的高级应用。第 4 篇介绍了 JFace 框架的知识及其应用。第 5 篇介绍了最新的 RCP 应用程序的开发。

本书结构清晰,注重实用,深入浅出,非常适合 Eclipse 开发人员学习使用,尤其适合 SWT/JFace 开发人员、Eclipse 插件开发人员和 RCP 应用程序开发人员。

本书封面贴有清华大学出版社防伪标签,无标签者不得销售。

版权所有,侵权必究。侵权举报电话:010-62782989 13501256678 13801310933

图书在版编目(CIP)数据

Eclipse SWT/JFace 核心应用/强锋科技编著. —北京:清华大学出版社,2007.3
(Java 开发利器)

ISBN 978-7-302-14717-6

I. E… II. 强… III. 软件工具-程序设计 IV. TP311.56

中国版本图书馆 CIP 数据核字(2007)第 022338 号

责任编辑:欧振旭 马子杰

封面设计:范华明

版式设计:赵丽娜

责任校对:王 云

责任印制:王秀菊

出版发行:清华大学出版社

<http://www.tup.com.cn>

c-service@tup.tsinghua.edu.cn

社 总 机:010-62770175

投稿咨询:010-62772015

地 址:北京清华大学学研大厦 A 座

邮 编:100084

邮购热线:010-62786544

客户服务:010-62776969

印 刷 者:北京鑫丰华彩印有限公司

装 订 者:三河市新茂装订有限公司

经 销:全国新华书店

开 本:185×260 印张:40.75 字数:904 千字

附光盘 1 张

版 次:2007 年 3 月第 1 版 印 次:2007 年 3 月第 1 次印刷

印 数:1~6000

定 价:69.00 元

本书如存在文字不清、漏印、缺页、倒页、脱页等印装质量问题,请与清华大学出版社出版部联系调换。联系电话:(010)62770177 转 3103 产品编号:019250-01

敬请关注“Java开发利器”之
Eclipse 三剑客下列热门图书
的出版!

《Eclipse 从入门到精通 (第2版)》

本书为备受读者推崇的程序类经典畅销书《Eclipse 从入门到精通》的全新升级版。本书采用Eclipse 3.2版写作。在原书的基础上,本书重写了所有的API,增加了RCP 程序开发、Form 组件、ActiveX 访问、JFace 数据绑定等内容。本书所涉及的其他相关软件包都升级成了最新版本。本书是每一个学习 Eclipse 程序开发者的绝佳选择。

本书作者陈刚有多年的开发经验和项目经验,其写作认真、严谨,对读者非常负责。作者从事过相关技术培训,能够很好地把握读者学习的重点和难点。

《Eclipse Web 开发从入门到精通(实例版)》

本书为清华大学出版社“Java开发利器”系列中的Eclipse Web 开发分册。本书内容丰富,实例典型,涉及Eclipse 与 Ant、Hibernate、JUnit、JBoss、Struts、CVS 等多种开源项目的整合使用。全书以实例驱动,通过代码掌握Web 开发,并特别提供了学生成绩管理系统、留言板、图书管理系统、光盘资料管理及网上书店管理系统等案例,适合相关毕业设计和项目开发人员学习。



清华大学出版社长期以来致力于 Java 开发技术专业书籍的编撰和出版，所出版的 Java 技术书籍以其系统、专业、实用而深受广大读者的喜爱。本套“Java 开发利器”丛书也秉承了清华程序设计图书的传统优势，集技术与应用于一体，将全面提升您的 Java 应用开发水准，并将 Java 开发的前沿技术一网打尽。

第一批（已出版）：

《Eclipse 从入门到精通》
《Hibernate 开发及整合应用大全》
《Struts Web 设计与开发大全》
《Tomcat Web 开发及整合应用》
《J2ME 手机游戏开发技术详解》
《JSP 网络开发技术与整合应用》
《Spring 从入门到精通》

第二批：

《Eclipse 开发从入门到精通（第2版）》
《Eclipse SWT/JFace 核心应用》
《Eclipse Web 开发从入门到精通（实例版）》
《Ant 开发及整合应用详解》

前言

从 Java 诞生至今，已经在太多的领域取得了成功，然而它却很少在图形界面程序上崭露头角。究其原因，Java 语言默认的图形界面开发包 AWT 和 Swing 实在是难逃其咎，无论是速度还是外观，它们都难以让人接受。如今，Eclipse 组织编写的 SWT 开发包，为 Java 程序员提供了 AWT 和 Swing 之外的一个更佳的选择。

为了帮助读者全面地了解 SWT 的知识体系，笔者精心编著了本书。本书依照读者的学习规律，首先介绍 SWT 的基本概念和各种基本控件的使用，在读者掌握了这些基本概念的基础上，再深入地讲解 JFace 和 RCP 的应用。整本书严格遵循由浅入深，循序渐进的原则，并结合大量的代码实例来讲述。

本书内容

本书共分为 5 篇，在内容的组织和编排上进行了精心的安排。

第 1 篇：SWT 起步篇（第 1~3 章）。首先介绍了 Java GUI 发展的历史、SWT 产生的背景，以及 SWT 与 JFace 和 Eclipse 的关系等，使读者对 SWT 的知识有一个整体的把握。然后介绍了 SWT 环境的配置，包括 Eclipse 的下载与安装、SWT 开发包的下载，以及可视化开发 SWT 的 Visual Editor 插件的下载与安装等。最后，对 Eclipse 的开发环境作了简单的介绍，使读者能快速地适应 Eclipse 的开发环境。

第 2 篇：SWT 进阶篇（第 4~8 章）。首先以一个简单的“Hello World”SWT 程序为例，介绍了 SWT 程序的基本结构。然后介绍 SWT 最常用的一些控件，如按钮、文本框、下拉框和列表框等。之后，介绍 SWT 中面板控件及面板所涉及的布局知识。最后详细讲述 SWT 中的事件模型。

第 3 篇：SWT 高级篇（第 9~13 章）。内容包括一些高级控件（如菜单、工具栏、表格、树、格式化文本等控件）的使用、SWT 的拖放支持、SWT 中的线程处理、SWT 中资源的处理和其他 SWT 的高级应用，包括打印支持、嵌入 ActiveX 控件等。通过了解这部分内容，读者将对 SWT 的知识有更加深刻的理解。

第 4 篇：JFace 篇（第 14~21 章）。首先介绍 JFace 与 SWT 开发的不同之处及 JFace 与 SWT 的关系。然后分别介绍 JFace 增强了 SWT 的各个方面，包括应用程序的窗口、各种对话框、向导对话框、首选项的应用、MVC 模式的各种控件、JFace 的一些工具类等。最后以一个 JavaScript 代码编辑器程序为例剖析 JFace 中有关对文本处理部分的操作。通过学习这部分内容，读者将进一步了解 JFace 的体系构架，为开发 RCP 程序打下基础。

第 5 篇：RCP 应用篇（第 22~23 章）。首先介绍 RCP 产生的背景以及 RCP 应用的前景，然后介绍如何开发一个 RCP 应用程序，之后详细介绍开发 RCP 常用的一些功能，最后介绍如何将 RCP 产品进行打包和发布，交付给用户使用。

本书特色

1. 基于理论，注重实践

本书对知识点的讲述都是从代码出发，然后基于代码，深入剖析所涉及的理论知识。这样做的好处是，避免了大量的理论罗列，而是先使读者有了一定的感性认识后，再继续深入，上升到理性，符合学习知识的客观过程。

2. 取材广泛，内容充实

本书覆盖 SWT 技术的各个方面，从最基本的简单概念到 SWT 线程的高级知识，再到 SWT 的扩展 JFace，然后到 RCP 的应用。本书内容涵盖 SWT 的各个方面，是对 SWT 内容体系的整体总结。

3. 示例典型，应用广泛

作者精心挑选了大量的示例程序，它们都是根据作者在实际开发中的经验总结而来，涵盖了在实际开发中所遇到的各种问题。而且有些程序能够直接拿到项目中使用，避免了读者进行二次开发。

4. 注重基础，高于基础

本书不仅对基础知识进行了讲解，而且还大量讲述了所涉及的一些模式，不仅告诉读者如何做，而且告诉读者为什么这样做。这样使读者不仅知道如何应用，而且能理解这样应用的理论是什么，从而便于读者以后更深入地学习。

5. 详细探讨了 RCP 的相关知识

RCP 是 Eclipse 的一大亮点，目前市场上还没有全面介绍其使用的书籍。本书对 RCP 的相关知识进行了详细介绍。

6. 配多媒体视频演示光盘，加速学习

作者为读者制作了多媒体视频演示光盘，内容涵盖在开发代码时常用的操作，这使读者能快速地了解开发的整个过程。

读者对象

本书具有知识全面、实例精彩、指导性强的特点，力求以全面的知识性及丰富的实例来指导读者透彻地学习 SWT 的各方面技术。本书可以作为 SWT 的入门书籍，也可以帮助中级读者提高技能，对高级读者也有一定的启发意义。

本书适合以下人员阅读：

- ☐ Java 开发人员
- ☐ Eclipse 开源项目爱好者
- ☐ SWT/JFace 开发人员
- ☐ Eclipse 插件开发人员
- ☐ RCP 应用程序开发人员

□ 创作团队

本书由强锋科技组织编著，由那静主笔。其他参与编写资料整理和光盘制作的人员有郑耀东、蔡骞、李晓辉、吕静、张岫、王俊标、陈晨、李卓龙、高守传、郭瑞、周宇炜、蔡雪焄、陈杰、荣飞、郑林、张路平、项宇峰、罗皓菡、赵正坤、公芳亮、程明雷、梁文建、马斗、邱哲、宋昕、陈刚、张传毓、赖梅艳、杨晓强、梁计锋、强致懿、郭腊梅、肖萍、程鹏辉、吕静、张增强、贺广治等。衷心感谢参与本书的每一个人！

编著者



读者意见反馈卡

您购买的书名: _____ 您的姓名: _____ 性别: ☐男 ☐女
年龄: _____ 文化程度: _____ 职业: _____
邮编: _____ 通信地址: _____ E-mail: _____
您常用的软件: 1 _____ 2 _____ 3 _____ 4 _____

您购买本书的原因 (可多选):

☐封面与装帧 ☐引言目录 ☐正文内容 ☐丛书风格 ☐价格 ☐光盘 ☐专业性强 ☐别人介绍
☐出版社或作者名声 ☐售后服务

本书最令您满意的是 (可多选):

☐专业性强、覆盖面广 ☐内容翔实、定位准确 ☐精益求精、售后服务

您可以承受的图书价格:

☐20 元以下 ☐30 元以下 ☐40 元以下 ☐50 元以下 ☐只要内容好, 不论价格

您对本书的评价:

封面装帧: <input type="checkbox"/> 很好	<input type="checkbox"/> 较好	<input type="checkbox"/> 一般	<input type="checkbox"/> 不满意 建议_____
印刷质量: <input type="checkbox"/> 很好	<input type="checkbox"/> 较好	<input type="checkbox"/> 一般	<input type="checkbox"/> 不满意 建议_____
正文质量: <input type="checkbox"/> 很好	<input type="checkbox"/> 较好	<input type="checkbox"/> 一般	<input type="checkbox"/> 不满意 建议_____
写作风格: <input type="checkbox"/> 很好	<input type="checkbox"/> 较好	<input type="checkbox"/> 一般	<input type="checkbox"/> 不满意 建议_____
专业水平: <input type="checkbox"/> 很好	<input type="checkbox"/> 较好	<input type="checkbox"/> 一般	<input type="checkbox"/> 不满意 建议_____

您希望增加哪些图书选题: 1 _____ 2 _____ 3 _____

您认为本书有哪些错误:

章 _____ 节 _____ 页码 _____ 行 _____ 列 _____ 图号 _____ 错误 _____ 应改为 _____
章 _____ 节 _____ 页码 _____ 行 _____ 列 _____ 图号 _____ 错误 _____ 应改为 _____
章 _____ 节 _____ 页码 _____ 行 _____ 列 _____ 图号 _____ 错误 _____ 应改为 _____
章 _____ 节 _____ 页码 _____ 行 _____ 列 _____ 图号 _____ 错误 _____ 应改为 _____

您的其他建议:

1 _____
2 _____
3 _____

请填写好本卡后寄给:

清华大学校内金地公司

邮编: 100084

电话: (010) 62791976-220

《Eclipse SWT/JFace 核心应用》编辑部收

传真: (010) 62788903

公司网址: www.thjd.com.cn

E-mail: oyzx_sp@263.net

如需本书可与本编辑部联系邮购, 汇款请按以上地址填写, 另加邮费 15% (挂号)

目 录

第 1 篇 SWT 起步篇

第 1 章	Java 语言的 GUI 历史	2
1.1	最初的 AWT	2
1.2	Swing 工具包	3
1.3	Eclipse 的诞生	3
1.4	Eclipse 贡献 SWT 工具包	5
1.4.1	SWT 的结构	6
1.4.2	SWT 所支持的操作系统	6
1.5	Sun AWT/Swing 与 Eclipse SWT	7
1.5.1	Swing 与 SWT 的比较	7
1.5.2	SWT 的优势和不足	8
1.6	SWT 与 JFace、Eclipse	9
1.6.1	JFace 是 SWT 的扩展	9
1.6.2	Eclipse 的 UI 界面基于 JFace	10
1.7	本章小结	10
第 2 章	配置 SWT 开发环境	11
2.1	下载和安装 Eclipse	11
2.1.1	Eclipse 下载页面介绍	11
2.1.2	下载 Eclipse	12
2.1.3	安装 Eclipse 语言包	14
2.1.4	在不同的语言中切换	15
2.2	直接获取 SWT 工具包	16
2.3	下载和安装 Visual Editor	17
2.3.1	Visual Editor 的下载	17
2.3.2	Visual Editor 的安装	18
2.4	第一个 SWT 程序	19
2.4.1	创建 SWT 程序	19
2.4.2	编译和运行程序	20
2.5	本章小结	21
第 3 章	Eclipse 开发环境概述	22
3.1	Eclipse 界面一览	22

3.2	Eclipse 项目的文件结构	23
3.2.1	设置编译后.class 文件的保存目录	23
3.2.2	导入项目使用的包	25
3.2.3	设置编译方式	26
3.2.4	运行程序	27
3.3	常用的代码编辑功能	28
3.3.1	添加注释	28
3.3.2	自定义格式化代码	28
3.3.3	自动生成 getter 和 setter 代码	30
3.3.4	代码的重构	31
3.3.5	查看源代码	31
3.3.6	代码的展开和折叠	32
3.3.7	代码比较	33
3.3.8	子类中覆盖父类的方法	34
3.4	代码错误提示	34
3.4.1	如何定位错误	34
3.4.2	自动修正错误	35
3.5	文件查找	35
3.5.1	文件内部查找	35
3.5.2	项目内查找	36
3.6	使用快捷键	36
3.6.1	显示快捷键说明	37
3.6.2	自定义快捷键	37
3.7	本章小结	38

第 2 篇 SWT 进阶篇

第 4 章	SWT 开发基础	40
4.1	SWT 应用程序基本结构	40
4.2	Display 类	42
4.2.1	Display 类概述	42
4.2.2	Display 类常用方法	42
4.3	Shell 类	45
4.3.1	Shell 类概述	45
4.3.2	不同窗口的样式	46
4.3.3	应用多个样式	47
4.3.4	Shell 类的主要方法	47
4.3.5	创建多个窗口	49

4.4	SWT 包类结构	51
4.5	本章小结	52
第 5 章	SWT 基本组件	53
5.1	SWT 控件类概述	53
5.1.1	窗口小部件: Widget	53
5.1.2	Widget 的继承关系	53
5.1.3	SWT 中的子类	54
5.1.4	控件 (Controls) 与面板 (Composites)	55
5.1.5	Widgets 不是 Controls	55
5.2	按钮 (Button)	58
5.2.1	普通按钮 (SWT.PUSH)	58
5.2.2	切换按钮 (SWT.TOGGLE)	59
5.2.3	箭头按钮 (SWT.ARROW)	60
5.2.4	单选按钮 (SWT.RADIO)	60
5.2.5	多选按钮 (SWT.CHECK)	61
5.2.6	常用的方法	63
5.3	标签 (Label)	64
5.3.1	文本标签	64
5.3.2	分割线标签	64
5.3.3	自定义标签 (CLabel)	65
5.4	文本框 (Text)	65
5.4.1	文本框的样式	66
5.4.2	文本框程序示例	66
5.4.3	常用的方法	68
5.5	列表框 (List)	69
5.5.1	列表框的样式	69
5.5.2	列表框程序示例	70
5.5.3	常用的方法	73
5.6	组合框 (Combo)	74
5.6.1	组合框的样式	75
5.6.2	组合框程序示例	75
5.6.3	组合框的常用方法	77
5.6.4	自定义组合框 CCombo 类	77
5.7	本章小结	78
第 6 章	面板容器类	79
6.1	面板类 (Composite)	79
6.1.1	面板类的样式	79

6.1.2 面板类的常用方法	80
6.2 分组框 (Group)	80
6.3 选项卡 (TabFolder)	81
6.3.1 选项卡的基本构成	81
6.3.2 设置底部显示选项卡	82
6.3.3 设置选项卡图标	82
6.3.4 选项卡的常用方法	83
6.4 自定义选项卡 (CTabFolder)	83
6.4.1 带有“关闭”按钮的选项卡	84
6.4.2 带有边框的选项卡	85
6.4.3 显示“最大化/最小化”按钮	85
6.4.4 设置选项卡的颜色和背景图片	85
6.4.5 仿 Eclipse 编辑区的选项卡	87
6.4.6 限制选项卡文字的长度	90
6.4.7 设置右上角控件	91
6.4.8 自定义选项的常用方法	91
6.5 分割窗框 (SashForm)	92
6.5.1 分割窗框的样式	92
6.5.2 设置窗框显示的比例	93
6.5.3 设置窗框最大化所显示的控件	93
6.6 自定义分割框 (CBanner)	94
6.6.1 改变分割线的外观	95
6.6.2 Eclipse 中的 CBanner	95
6.7 滚动面板 (ScrolledComposite)	95
6.7.1 设置滚动条的样式	96
6.7.2 滚动面板的其他方法	97
6.8 本章小结	97
第 7 章 SWT 布局管理器	98
7.1 布局管理器概述	98
7.1.1 绝对定位	98
7.1.2 托管定位	98
7.1.3 常见的布局管理器	100
7.2 FillLayout (充满式布局)	100
7.2.1 水平填充 (默认) 和垂直填充	101
7.2.2 设置四周补白	102
7.3 RowLayout (行列式布局)	102
7.3.1 设置折行显示: wrap 属性	103
7.3.2 设置空间大小: pack 属性	103

7.3.3	设置填充方式: type 属性	103
7.3.4	设置是否充满整行: justify 属性	104
7.3.5	设置补白和间隔	104
7.3.6	设置控件的大小 RowData	105
7.3.7	设置是否等宽或等高: fill 属性	105
7.4	GridLayout (网格式布局)	106
7.4.1	设置网格的列数: numColumns 属性	106
7.4.2	设置网格等宽: makeColumnsEqualWidth 属性	107
7.4.3	设置补白和间隔	107
7.4.4	使用 GridData 对象	107
7.4.5	设置单元格对齐方式: horizontalAlignment 和 verticalAlignment 属性	108
7.4.6	设置缩进大小: horizontalIndent 和 verticalIndent 属性	109
7.4.7	设置单元格跨行和跨列显示: horizontalSpan 和 verticalSpan 属性	109
7.4.8	设置单元格空间的抢占方式: grabExcessHorizontalSpace 和 grabExcessVerticalSpace 属性	110
7.4.9	设置的控件大小: minimumWidth 和 minimumHeight 属性	111
7.4.10	设置控件大小: widthHint 和 heightHint 属性	111
7.4.11	样式常量对照表	112
7.5	FormLayout (表格式布局)	112
7.5.1	设置补白和间隔	113
7.5.2	使用 FormData 对象	113
7.5.3	使用 FormAttachment 对象	114
7.5.4	设置控件的相对位置	115
7.6	StackLayout (堆栈式布局)	115
7.7	自定义布局管理器	117
7.7.1	布局的基本原理	117
7.7.2	布局计算的常用方法	118
7.7.3	自定义布局类 (BorderLayout)	119
7.8	使用 VE 可视化布局	123
7.8.1	创建可视化的类	123
7.8.2	进行布局设置	124
7.9	本章小结	125
第 8 章	SWT 中的事件模型	126
8.1	事件模型概述	126
8.1.1	监听器 (Listener)	126
8.1.2	事件 (Event)	127
8.1.3	注册监听器	127
8.1.4	适配器	128

8.1.5 常见的事件	128
8.2 事件处理的常用写法	130
8.2.1 内部匿名类	130
8.2.2 内部类	130
8.2.3 实现接口的类	131
8.2.4 继承的类的方法	132
8.3 键盘事件	132
8.3.1 键盘事件程序示例	132
8.3.2 键盘事件的各种属性	134
8.4 鼠标事件	136
8.4.1 鼠标事件程序示例	136
8.4.2 鼠标事件的各种属性	139
8.5 其他常用的事件	139
8.5.1 选中事件	139
8.5.2 文本修改程序示例	140
8.5.3 文本修改事件: VerifyEvent 的各种属性	142
8.5.4 文本修改事件: VerifyEvent 和 ModifyEvent 的区别	143
8.6 无类型的事件	144
8.6.1 注册无类型事件监听器	144
8.6.2 无类型事件程序示例	145
8.7 本章小结	146

第 3 篇 SWT 高级篇

第 9 章 SWT 高级控件	148
9.1 链接文本 (Link)	148
9.2 菜单 (Menu 和 MenuItem)	149
9.2.1 菜单与菜单项之间的关系	150
9.2.2 菜单的样式	151
9.2.3 菜单项的样式	152
9.2.4 设置菜单项的图标	152
9.2.5 设置菜单项快捷键	152
9.3 工具栏 (ToolBar 和 ToolItem)	153
9.3.1 工具栏图片资源的管理	155
9.3.2 工具栏的不同样式	157
9.3.3 工具栏按钮的不同样式	158
9.3.4 工具栏常用的方法	160
9.4 可拖动的工具栏 (CoolBar 和 CoolItem)	161

9.4.1 带有下拉选项的工具栏.....	163
9.4.2 常用的方法	163
9.5 系统托盘 (Tray 和 TrayItem)	164
9.6 滑动组件	167
9.6.1 滑块 (Slider)	167
9.6.2 刻度条 (Scale)	168
9.6.3 微调按钮 (Spinner)	168
9.7 进度条 (ProgressBar)	169
9.8 对话框	170
9.8.1 消息提示框 (MessageBox)	171
9.8.2 文件目录对话框 (DirectoryDialog)	172
9.8.3 文件对话框 (FileDialog)	173
9.8.4 颜色对话框 (ColorDialog)	175
9.8.5 字体对话框 (FontDialog)	176
9.8.6 打印对话框 (PrintDialog)	177
9.9 表格 (Table、TableItem 和 TableColumn)	178
9.9.1 Table、TableItem 和 TableColumn 的关系	182
9.9.2 设置带有选择框的表格.....	182
9.9.3 设置可同时选中多行表格.....	183
9.9.4 可拖动的表格	184
9.9.5 设置单元格的图标	184
9.9.6 改变选中行高亮显示的颜色.....	185
9.9.7 带有上下文菜单的表格.....	186
9.9.8 可编辑的表格 (TableEditor)	187
9.9.9 用键盘控制表格 (TableCursor)	189
9.9.10 带有进度条的表格.....	191
9.9.11 表格小结	192
9.10 树 (Tree)	192
9.10.1 不同样式的树	193
9.10.2 为树添加图标	193
9.10.3 可编辑的树	196
9.10.4 表格树	197
9.10.5 树小结	198
9.11 格式化文本 (StyleText)	198
9.11.1 格式化对象 (StyleRange)	199
9.11.2 格式化文本的事件处理.....	200
9.11.3 对选中文本设置格式.....	201
9.11.4 自动为数字字符着色.....	203

9.11.5 换行自动设置背景颜色.....	204
9.12 浏览器	205
9.13 本章小结	210
第 10 章 SWT 中的拖放支持	211
10.1 可拖放的树	211
10.2 拖放原理概述	214
10.3 拖放源 (DragSource)	215
10.3.1 创建拖放源对象	215
10.3.2 定义拖放源数据传输类型.....	215
10.3.3 处理拖放源事件	216
10.4 拖放目标 (DragTarget)	218
10.4.1 定义目标对象	218
10.4.2 定义目标对象的数据传输类型.....	219
10.4.3 处理拖放目标事件.....	219
10.5 传输数据 (Transfer)	221
10.6 综合示例: 简单购物车	222
10.7 对剪贴板的操作	226
10.8 本章小结	229
第 11 章 SWT 线程.....	230
11.1 线程概述	230
11.1.1 什么是线程	230
11.1.2 创建线程的两种方式.....	231
11.2 SWT 中的 UI 线程.....	234
11.3 其他线程访问 UI 线程	234
11.4 改进的进度条	236
11.5 多线程程序设计	238
11.6 本章小结	243
第 12 章 SWT 系统资源	244
12.1 系统资源概述	244
12.1.1 什么是系统资源	244
12.1.2 释放资源的原则	245
12.1.3 访问资源的原则	246
12.1.4 何时释放资源	246
12.2 颜色 (Color)	247
12.2.1 系统颜色	248
12.2.2 RGB 颜色	249
12.3 字体 (Font)	250

12.4	光标 (Cursor)	251
12.5	图像 (Image)	252
12.5.1	画布类 (Canvas)	252
12.5.2	图像类 (Image)	254
12.5.3	图像数据类 (ImageData)	255
12.5.4	保存图像类 (ImageLoader)	256
12.5.5	Eclipse 的图标.....	256
12.6	SWT 绘图	257
12.6.1	使用绘制对象的方法.....	257
12.6.2	绘制线条	258
12.6.3	绘制字符	259
12.6.4	绘制填充图形	260
12.6.5	绘制图像	261
12.7	本章小结	261
第 13 章	SWT 的高级应用	262
13.1	打印支持	262
13.1.1	打印类 (Printer) 和打印数据类 (PrinterData)	262
13.1.2	打印程序示例概述.....	265
13.1.3	打印程序示例: 主窗口程序.....	265
13.1.4	打印程序示例: 打开文件程序.....	268
13.1.5	打印程序示例: 设置字体和颜色程序.....	268
13.1.6	打印程序示例: 打印文本的程序.....	269
13.1.7	打印程序示例: 打印文件后的效果预览.....	273
13.2	使用应用程序	274
13.3	对 AWT/Swing 程序的支持	275
13.4	OLE 和 ActiveX 控件的支持	275
13.4.1	OLE 控件的面板类 (OleFrame)	276
13.4.2	OLE 控件类 (OleClientSite 和 OleControlSite)	277
13.4.3	OLE 程序示例.....	278
13.5	Pocket PC 应用.....	280
13.6	Web 应用 SWT	282
13.7	本章小结	282

第 4 篇 JFace 篇

第 14 章	JFace 概述.....	284
14.1	配置 JFace 运行环境	284
14.2	第一个 JFace 程序	285

14.3	JFace 框架概述	287
14.4	JFace 的包结构	287
14.5	本章小结	289
第 15 章	应用程序窗口	290
15.1	JFace 的窗口类 (Window 类)	290
15.2	应用程序窗口 ApplicationWindow 类	292
15.3	带有菜单栏的主程序窗口	293
15.3.1	简单写字板程序示例	293
15.3.2	添加菜单栏的基本步骤	297
15.3.3	创建菜单项	297
15.3.4	菜单项的事件处理	298
15.4	带有工具栏的主程序窗口	299
15.5	带有状态栏的主程序窗口	300
15.6	其他处理事件的方法	304
15.6.1	“新建”操作	304
15.6.2	“保存”操作	305
15.6.3	“另存为”操作	306
15.6.4	“复制”、“剪切”和“粘贴”操作	307
15.7	本章小结	310
第 16 章	JFace 对话框	311
16.1	JFace 对话框概述	311
16.2	信息提示对话框 (MessageDialog)	312
16.2.1	创建信息提示对话框	312
16.2.2	错误消息对话框	314
16.2.3	确认消息对话框	315
16.2.4	消息对话框	315
16.2.5	询问对话框	316
16.2.6	警告对话框	316
16.2.7	JFace 的本地化	317
16.3	输入对话框 (InputDialog)	318
16.3.1	创建输入对话框	319
16.3.2	创建输入文本的验证类	320
16.4	带有提示信息的对话框 (TitleAreaDialog)	321
16.5	错误提示对话框 (ErrorDialog)	325
16.5.1	创建错误提示对话框	325
16.5.2	使用错误状态对象	327
16.5.3	同时显示多个错误信息	327

16.6 带有进度条的对话框 (ProgressMonitorDialog)	329
16.7 自定义对话框	330
16.7.1 自定义对话框程序示例	331
16.7.2 自定义对话框的步骤	333
16.8 本章小结	334
第 17 章 向导式对话框	335
17.1 向导式对话框概述	335
17.1.1 向导式对话框所涉及的类	335
17.1.2 向导式对话框的常用方法	336
17.2 简单的向导式对话框示例	338
17.2.1 第一个问题向导页面	338
17.2.2 第二个问题向导页面	339
17.2.3 感谢向导页面	340
17.2.4 创建向导	341
17.2.5 创建测试程序	342
17.3 保存对话框状态	344
17.4 复杂的向导式对话框示例	345
17.4.1 自定义向导页面	346
17.4.2 为向导添加帮助	349
17.5 向导式对话框的事件处理	350
17.6 本章小结	350
第 18 章 首选项	351
18.1 首选项概述	351
18.2 保存首选项的设置	353
18.2.1 首选项值的设置和获取	353
18.2.2 保存首选项所涉及的事件	354
18.3 显示首选项页面	354
18.3.1 创建一个首选项页面	354
18.3.2 创建首选项页面所对应的节点	357
18.3.3 显示首选项对话框	358
18.4 创建树型的导航菜单	360
18.4.1 第一种方法	361
18.4.2 第二种方法	361
18.5 首选项的选项设置	361
18.5.1 字段编辑器概述	362
18.5.2 使用字段编辑器基本步骤	362
18.5.3 布尔型字段编辑器 (BooleanFieldEditor)	364

18.5.4	颜色字段编辑器 (ColorFieldEditor)	364
18.5.5	字体字段编辑器 (FontFieldEditor)	364
18.5.6	路径列表字段编辑器 (PathEditor)	365
18.5.7	单选分组字段编辑器 (RadioGroupFieldEditor)	365
18.5.8	刻度条字段编辑器 (ScaleFieldEditor)	366
18.5.9	整数型字段编辑器 (IntegerFieldEditor)	366
18.5.10	选择路径字段编辑器 (DirectoryFieldEditor)	366
18.5.11	选择文件字段编辑器 (FileFieldEditor)	366
18.6	自定义首选项页面	367
18.7	首选项的事件处理	368
18.8	本章小结	369
第 19 章	MVC 的表格、树和列表	370
19.1	MVC 概述	370
19.2	表格 (TableViewer)	371
19.2.1	创建表格控制器 (Controller)	371
19.2.2	创建表格模型 (Model)	373
19.2.3	创建组织表格视图 (View)	375
19.2.4	添加和删除数据	376
19.2.5	增加表格排序功能	378
19.2.6	增加表格过滤功能	380
19.2.7	编辑表格单元	381
19.2.8	表格的事件处理	383
19.2.9	带有复选框表格 (CheckBoxTableViewer)	383
19.3	树 (TreeViewer)	384
19.3.1	树的基本性质	385
19.3.2	创建树 (TreeViewer)	386
19.3.3	对树的操作	389
19.4	树和表格的综合示例	390
19.4.1	文件浏览器功能概述	391
19.4.2	程序的整体框架	391
19.4.3	初始化树	393
19.4.4	初始化表格	394
19.4.5	程序的事件处理	396
19.5	列表 ListView	397
19.6	本章小结	398
第 20 章	JFace 的工具类	399
20.1	JFace 资源管理机制	399

20.1.1	图像描述符 (ImageDescriptor)	399
20.1.2	图像注册器 (ImageRegistry)	401
20.1.3	字体描述符和字体注册器	403
20.1.4	颜色描述符和颜色注册器	404
20.1.5	JFace 的资源管理器 (JFaceResources)	406
20.1.6	字符转换工具类 (StringConverter)	407
20.2	类型检查的工具类	407
20.3	本章小结	408
第 21 章	文本处理	409
21.1	文本处理概述	409
21.2	项目实战: JavaScript 编辑器	409
21.2.1	主窗口预览	409
21.2.2	项目文件结构	410
21.3	主窗口模块	411
21.3.1	代码实现	411
21.3.2	主窗口程序代码分析	414
21.3.3	启动主窗口程序	416
21.4	代码着色	417
21.4.1	源代码配置类 (SourceViewerConfiguration)	417
21.4.2	基于规则的代码扫描器类 (RuleBasedScanner)	419
21.4.3	设置代码扫描规则	420
21.4.4	提取类 (Token) 和文本属性类 (TextAttribute)	423
21.5	内容辅助	423
21.5.1	配置编辑器的内容助手	424
21.5.2	内容辅助类	424
21.5.3	辅助建议类 (CompletionProposal)	426
21.6	文档的撤销与重复	427
21.6.1	文档管理器对象 (DefaultUndoManager)	427
21.6.2	撤销操作的实现	427
21.6.3	恢复操作的实现	428
21.7	查找与替换窗口	429
21.7.1	窗口的界面设计	429
21.7.2	查找功能的实现	433
21.7.3	替换功能的实现	434
21.8	首选项的对话框	434
21.8.1	首选项页面的代码实现	435
21.8.2	打开首选项页面的代码	436
21.9	文件的打开、保存与打印	437

21.9.1 打开文件	437
21.9.2 保存文件	437
21.9.3 打印文件	438
21.10 帮助对话框	439
21.11 其他的一些工具类	440
21.11.1 事件管理类	440
21.11.2 资源管理类	441
21.11.3 程序中使用的常量	443
21.12 本章小结	444

第 5 篇 RCP 应用篇

第 22 章 富客户端平台 (RCP) 应用	446
22.1 RCP 概述	446
22.1.1 什么是 RCP	446
22.1.2 RCP 应用的现状	447
22.2 第一个 RCP 项目	448
22.2.1 创建插件项目	449
22.2.2 运行 RCP 程序	450
22.2.3 插件的文件清单	451
22.2.4 MANIFEST.MF 文件	452
22.2.5 build.properties 文件	454
22.2.6 plugin.xml 文件	454
22.3 RCP 运行的基本原理	455
22.3.1 插件类 MyRCPPugin	455
22.3.2 应用程序类 Application	456
22.3.3 工作台窗口类	458
22.3.4 操作类	459
22.3.5 透视图类	460
22.4 创建扩展的基本方法	460
22.4.1 使用扩展向导创建	460
22.4.2 手工创建	462
22.4.3 获取扩展点的帮助	463
22.5 本章小结	464
第 23 章 RCP 开发	465
23.1 扩展操作集 (Action Set)	465
23.1.1 操作集扩展点	465
23.1.2 编写代码创建操作对象	469

23.1.3 编写代码创建操作的步骤.....	471
23.1.4 其他与操作有关的扩展点.....	473
23.2 扩展视图	473
23.2.1 视图扩展点	474
23.2.2 视图类	475
23.2.3 视图之间的交互	477
23.2.4 添加视图的工具栏.....	480
23.2.5 添加上下文菜单	481
23.3 扩展编辑器	484
23.3.1 编辑器扩展点	485
23.3.2 编辑器类	485
23.3.3 打开编辑器	488
23.3.4 添加编辑器的菜单和工具栏.....	491
23.3.5 多页编辑器	493
23.4 透视图	495
23.4.1 透视图的布局	495
23.4.2 透视图扩展点	498
23.4.3 透视图类	499
23.5 首选项	500
23.5.1 首选项扩展点	500
23.5.2 首选项页面扩展点.....	501
23.6 帮助文档	503
23.6.1 联机帮助文档扩展点.....	504
23.6.2 扩展配置	505
23.6.3 联机帮助的目录结构.....	506
23.6.4 添加动态帮助	507
23.7 RCP 产品的发布	508
23.7.1 Eclipse 产品配置.....	508
23.7.2 导出 RCP 产品	510
23.7.3 运行 RCP 产品	510
23.8 本章小结	511
第 24 章 Eclipse 表单	512
24.1 Eclipse 表单概述.....	512
24.1.1 什么是 Eclipse 表单	512
24.1.2 Eclipse 表单的特性.....	513
24.1.3 Eclipse 表单使用的类包.....	513
24.2 表单开发基础	513
24.2.1 视图中使用表单	513

24.2.2	多页编辑器中使用表单.....	515
24.2.3	SWT 程序中使用表单.....	518
24.2.4	获得表单工具对象 (FormToolkit) 一般方法	519
24.3	表单的各种控件	519
24.3.1	可滚动的表单 (ScrolledForm)	519
24.3.2	可折叠的面板 (ExpandableComposite)	520
24.3.3	内容区 (Section)	523
24.3.4	超链接 (Hyperlink)	525
24.3.5	表单文本 (FormText)	527
24.4	表单的布局管理器	531
24.4.1	表格布局 (TableWrapLayout)	531
24.4.2	列布局 (ColumnLayout)	534
24.5	表单的高级应用	536
24.5.1	Master/Details 模式.....	536
24.5.2	实现 Master/Detail 示例程序.....	537
24.6	本章小结	544
第 25 章	项目实战——客户关系管理系统	545
25.1	系统概述	545
25.1.1	系统预览	545
25.1.2	基本概念介绍	546
25.1.3	系统的运行环境	547
25.1.4	系统文件结构的说明.....	547
25.2	UI 界面设计	547
25.3	业务层设计	549
25.3.1	业务层服务的定义.....	549
25.3.2	业务层的实现	551
25.3.3	业务层服务的管理.....	552
25.3.4	业务层 UML 图.....	553
25.3.5	如何调用业务对象.....	554
25.4	数据库层设计	555
25.4.1	数据库接口类	556
25.4.2	实现了 MySQL 数据库类	556
25.4.3	如何调用数据访问对象.....	561
25.4.4	应用多种数据库	562
25.4.5	数据库的初始化的脚本.....	563
25.4.6	表所对应的 POJO 类.....	564
25.5	登录模块	566
25.5.1	系统的上下文对象保存登录状态.....	566

25.5.2	登录验证的实现	567
25.5.3	登录窗口的实现	569
25.6	主窗口界面	572
25.6.1	工作台的实现	572
25.6.2	系统托盘的实现	573
25.6.3	菜单栏和工具栏的实现	575
25.6.4	操作管理类 (ActionManager)	579
25.6.5	新建客户操作 (NewCustomerAction)	580
25.6.6	打开视图操作 (ShowViewAction)	581
25.7	各种视图和编辑器的实现	582
25.7.1	快速新建客户视图	583
25.7.2	客户列表视图	585
25.7.3	客户详细编辑器	590
25.7.4	联系人列表视图	595
25.7.5	快速新建联系人视图	597
25.7.6	搜索视图	600
25.7.7	导航视图	603
25.8	新建客户联系人向导	609
25.8.1	新建客户向导	609
25.8.2	新建联系人向导	615
25.9	首选项的实现	618
25.10	plugin.xml 文件清单	623
25.11	本章小结	626



第 1 篇



SWT 起步篇

第 1 章 Java 语言的 GUI 历史

第 2 章 配置 SWT 开发环境

第 3 章 Eclipse 开发环境概述



第 1 章 Java 语言的 GUI 历史

对于图形化操作系统来说，GUI（Graphical User Interface，图形用户接口）是最重要的组成部分。GUI 是一组计算机接口，在传统的操作系统 MS-DOS 文字模式下，屏幕上显示的是单调的文字接口，使用者必须通过键盘输入指令才能操作计算机。GUI 的操作环境以图形、图标及窗口方式显示，用户可以通过一个焦点选择工具，如用鼠标来进行操作。GUI 接口的亲和性设计可以说是操作系统设计上的一大突破。随着操作系统向图形化方向的发展，各种编程语言也随之纷纷实现 GUI 接口，支持用户的 GUI 编程。基于本书的主题，我们着重探讨 Java 语言的 GUI 发展史。

1.1 最初的 AWT

1995 年，SUN 公司面向全球发布了对计算机语言发展具有划时代意义的 Java 语言。Java 惊世骇俗的宣言——“一次编写，到处运行（write once, run anywhere）”从此也深深地留在了开发人员的心中。“一次编写，到处运行”的特性不仅适用于其他 Java 组件，同样也适用于它所包含的名为 AWT（Abstract Window Toolkit）的库，用来构建图形用户接口应用程序。它曾许诺：一个具有下拉菜单、命令按钮、滚动条以及其他常见的 GUI 控件的应用程序将能够在各种操作系统上运行而不必重新编译成针对某一平台的二进制代码，包括 Microsoft Windows、Sun Solaris、Apple Mac OS 以及 Linux。

但是 AWT 有一个致命弱点：功能很弱。AWT 必须使用所有图形操作系统的图形接口功能的交集，因为 AWT 的界面只有一套。所以，为了保证移植性，就只能使用所有系统都能够支持的最少特性。因而我们经常可以听见有人抱怨 AWT 的功能太少，图形太难看等，这是为了保证移植性而作出的牺牲。

所以在随后的应用过程中发现的事实情况是：在图形用户接口（GUI）方面，Java 一直无法与 C++、Power Builder、Visual Basic、Delphi 之类的语言相抗衡，而使用早期 Java/AWT 包所开发的接口实在是让人不敢恭维。如图 1.1 所示就是一个典型的 AWT 文件选择窗口应用。

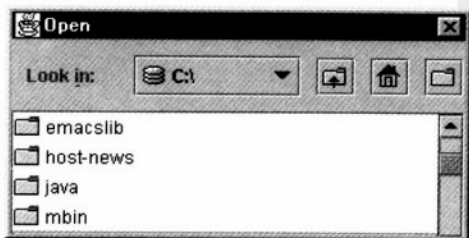


图 1.1 AWT 文件选择窗口

用 AWT 开发的应用程序既缺少流行 GUI 程序的许多特性，又不能达到在显示和行为上像用本地窗口构建库开发的程序一样的目标，因此应该有一个更好的库来让 Java GUI 取得成功。

1.2 Swing 工具包

基于 AWT 在 GUI 方向上失败的这一事实，SUN 公司最终决定放弃 AWT 项目。随后在 1997 年的 JavaOne 大会上提出，并在 1998 年 5 月发布的 JFC (Java Foundation Classes) 中包含了一个新的使用 Java 窗口的开发包。这个新的 GUI 组件叫做 Swing，从结构上看，感觉它是对 AWT 的升级，并且看起来对 Java 占据计算机世界很有帮助。

遗憾的是，在以后的应用中，Swing 依然受到了开发人员的抱怨，遭到很多软件开发商的质疑。尽管 Swing 在 AWT 的基础上作出了巨大的改进，但它仍然没能使 Java 成为构建桌面应用程序的优秀工具。它看起来是那样的庞大，虽然官方做了很多宣传与解释，但是没有开发人员会认为 Swing 是轻量级的。事实上，Swing 是一个非常巨大的 GUI 库，这一点已经是大家的共识，尤其对于初学者来说，很难理清其复杂的结构。

综合来说，Swing 应用程序不像本地应用程序一样执行，外观也不一样。Java 要想摆脱目前的这种局面，它的 GUI 仍需要改进。如图 1.2 所示就是一个典型的 Swing 应用。

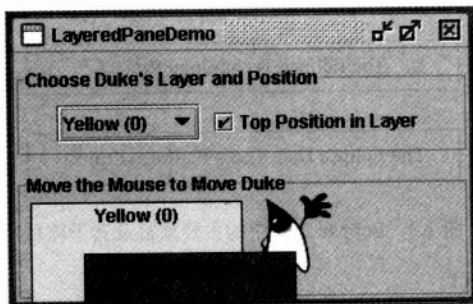


图 1.2 Swing 应用

1.3 Eclipse 的诞生

在进入 2000 年以后，为了对抗微软越来越强大的垄断地位，IBM 表示将投入 10 亿美元开发 Linux 产品，包括 PC、笔记本电脑、服务器和大型计算机。在一系列的举措中，最后被事实证明影响最深远的应该就是 Eclipse 计划。在 2001 年 6 月，IBM 宣布捐赠价值 4000 万美元的软件工具作为公共财产，并成立一个针对开发人员所设置的开放源代码机构。

Eclipse 平台完全基于 Java 编写而成，因此也具有跨平台的特性，可以在 Linux 和 Windows 平台下共同使用，即同样的代码不加修改即可在两个操作系统下顺利运行。这样，IBM 实际上拥有了全部的开放源代码程序员为它服务，不管是 Windows 的还是 Linux 的。

同时，也促进了开源事业的发展，这确实确实是商业软件公司在策略上的一次进步。

Eclipse 项目由协会的管理委员会（包括每个协会成员公司的一名代表）管理并主持。这个委员会制定项目目标和宗旨，在两个主导宗旨之间寻找平衡：培养一个健康的开源小区以及为成员创造商业机会。在运作上，Eclipse 项目作为一个整体由对象管理委员会（Project Management Committee, PMC）管理。在目前的设计中，Eclipse 主要由如图 1.3 所示的几个项目组成，每一个项目都遵照 CPL1.0 协议发布。

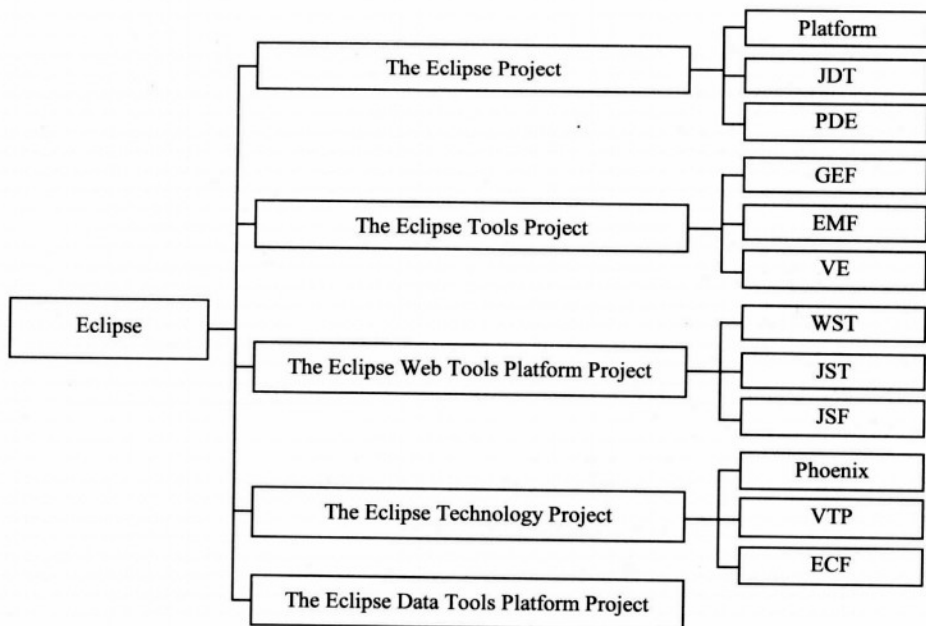


图 1.3 Eclipse 工程中的主要项目及其子项目

1. The Eclipse Project

The Eclipse Project 项目打造功能完善并且完全免费开源的工业级开发平台，具有很高的工具聚合性。在该项目下有 3 个重要的子项目。

- ❑ **Eclipse Platform Project:** 是 Eclipse 的核心框架产品，为所有插件和服务创建构件模型并提供运行环境，提供的框架和服务包含工作台的用户接口实现与本地部件调用服务。提供底层的系统资源管理与插件生命周期管理。
- ❑ **Eclipse JDT (Java Development Tools) Project:** 是一个非常优秀的 Java 开发工具集，在 Java 工程开发过程中提供非常丰富的功能支持开发。它本身以插件形式存在于 Eclipse Platform 中。
- ❑ **Eclipse PDE (Plug-in Development Environment) Project:** 提供了很多的工具来帮助开发人员基于 Eclipse 编写插件，使得插件在调试、运行期非常方便。

2. The Eclipse Tools Project

The Eclipse Tools Project 项目使更多的开源和非开源的工具作为插件支持 Eclipse 平台, 这个项目的存在可以协调各方开发人员共同开发某些功能的插件, 避免重复开发并确保各个工具之间可以良好地集成。在该项目下有 3 个重要的子项目。

- ❑ **Graphical Editor Framework (GEF)**: 允许开发人员为已有的应用创建一个富图形编辑器接口。GEF 运用基于 SWT 绘图插件 Draw2d 在 Eclipse 中创建一个图形环境。用户可以利用 GEF 提供的公共方法或者在特殊的领域下扩展它们。GEF 使用能简洁地改变应用于视图模型的 MVC 架构。GEF 是一个能为大多数程序提供链接和构造基础的完整应用。包括但不限于: 流构造器、图形接口构造器、UML 图表编辑器 (例如流程图和类图) 及类似于 HTML 的所见即所得的文本编辑器。
- ❑ **Eclipse Modeling Framework (EMF)**: 是一个建模框架和基于结构化数据模型的代码生成工具。它能够将按照约定所进行描述的 XML 和其他 EMF 工具产生或运行时支持一套 Java 类模型, 以及用于查看和处理模型操作的接口类。这套模型可以用包含注释的 Java 代码、XML 文档或者用 ROSE 之类的建模工具所制作的模型导出成 EMF。最重要的是, EMF 提供其他基于 EMF 的工具及应用程序的基础协作平台。
- ❑ **Eclipse Visual Editor (VE)**: 是 Eclipse 为插件开发者提供的可视化的 GUI 部件编辑器, 可以通过拖动的方式任意地摆放和规划部件的布局。

3. The Eclipse Web Tools Platform Project

The Eclipse Web Tools Platform Project 项目提供了一套通用并且具有高扩展性的 Web 开发平台, 该平台符合现有 J2EE 及其相关标准。平台中包含了很多支持开发 J2EE Web 应用的工具。在目前阶段主要关注基于标准的 Web 工具和 Java 运行时的环境。

4. The Eclipse Technology Project

The Eclipse Technology Project 项目的任务是为那些开源项目开发人员以及自由学者等提供一个参与到 Eclipse 发展进程中的管道, 使得他们在专注于自己领域的同时, 也可以融入到 Eclipse 的发展中来。

5. The Eclipse Data Tools Platform Project

The Eclipse Data Tools Platform Project 项目为 Eclipse 平台提供了一个全新的、对数据库技术提供各种功能的工具。

1.4 Eclipse 贡献 SWT 工具包

在 Eclipse 发布包中所包含的众多组件里, SWT (Standard Widget Toolkit) 无疑是一个巨大的亮点, 它的出现极大地带动了 Java 在 GUI 开发领域中的发展。SWT 是 IBM 的一个专门的部门独立进行研发的, 该部门最初的目的是想创建一套 AWT (Abstract Window Toolkit) 与 Swing 的替代品, 因此对于开发 SWT 所制定的目标是创建一套 GUI 工具, 使得

基于此开发出来的应用在外观上能够和本地操作系统的本地部件那样精美并且性能优异。

1.4.1 SWT 的结构

在 IBM 开发 Eclipse 的过程中,开发人员们使用了一种新的模式来完成窗口部件的创建:将 SWT 的功能实现完全构筑在以 JNI 为基础,对运行平台的直接调用封装上。它提供了与平台无关的 API,该 API 与操作系统的本机窗口环境紧密地集成在一起。该工具箱使开发人员不必面对在使用 Java 的抽象窗口工具箱 (Abstract Window Toolkit, AWT) 或 Java 基础类 (Java Foundation Classes, JFC) 时在许多设计和实现方面所要做的权衡。

如图 1.4 所示为 SWT 的结构。

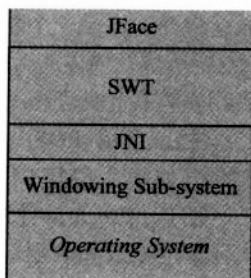


图 1.4 SWT 的结构

1.4.2 SWT 所支持的操作系统

SWT 的功能没有通过任何 Java 虚拟机来操作,而是直接调用 Windows GDI 和 Shell 功能。这一点是通过 JNI 方法调用完成。在 Eclipse 白皮书中对 SWT 的描述是:“由一套独立于操作系统的 API 以及本地原生窗口集成起来的图形库与小部件集”。这样做最大的好处是,可以使开发出来的应用具备本地操作系统的外观与特性,甚至包括 Pocket PC。只有在当前操作系统中找不到需要的部件时,SWT 才会自己绘制图形,这样做很明显可以使得应用的响应速度有很大提升。当然,SWT 的优点同时也是它的缺点,SWT 必须为每一种操作系统准备一套本地函数库,这一点就不像 Swing 那样灵活。不过目前 SWT 支持的操作系统基本上已经覆盖了常见的几种,如表 1.1 所示。

表 1.1 SWT 支持的操作系统列表

操 作 系 统	用 户 界 面
Microsoft Windows XP/2000/NT/98/Me	Windows
Microsoft Windows PocketPC 2002 Strong ARM	Windows
Microsoft Windows PocketPC 2002 Strong ARM (J2ME)	Windows
Red Hat Linux 9 x86	Motif, GTK 2.0

续表

操 作 系 统	用 户 界 面
SUSE Linux 8.2 x86	Motif, GTK 2.0
Other Linux x86	Motif, GTK 2.0
Sun Solaris 8 SPARC	Motif
IBM PowerPC	Motif
HP-UX 11i hp9000 PA-RISC	Motif
QNX x86	Photon
Mac OS	Carbon

1.5 Sun AWT/Swing 与 Eclipse SWT

从前面的介绍不难看出, 在当前的 Java GUI 领域中, SWT 无论是在性能还是外观上, 都超越了 SUN 公司所提供的 AWT 和 Swing。当然, SUN 公司对此非常不满。因为它们 Netbeans 与 Eclipse 做的是相同的事, 因此它们拒绝让 Eclipse 获得 Java 认证, 因为里面有源代码, 所以 Eclipse 产品必须很小心地使用单词“Java”这个 SUN 公司的商标。Eclipse 甚至不能把自己称为一个 Java IDE, SUN 公司已经威胁过要采取法律行动来制止 Eclipse 基金组织在任何时候把 Eclipse 称作一个 Java IDE。结果之一就是在 Eclipse 上创建的 GUI 设计工具, 允许构建 Swing/AWT GUI, 却不允许向里面拖放 SWT 窗口控件。

1.5.1 Swing 与 SWT 的比较

Swing 提供给开发人员一个抽象层次很高, 并且体积十分庞大的组件库。从学习以及使用上来说, 它确实不如 SWT 的 API 那样简单明了, 容易上手。但是如果需要完成一个非常复杂的 GUI 应用, Swing 可能会在某些方面显得更灵活一些。如表 1.2 所示, 可以清楚地对这 3 种 GUI 库所支持的部件作一个横向的对比。

表1.2 可视化部件对比

Component	SWT	Swing	AWT
Button	×	×	×
Advanced Button	×	×	
Label	×	×	×
List	×	×	×
Progress Bar	×	×	
Sash	×	×	
Scale	×	×	
Slider	×	×	

续表

Component	SWT	Swing	AWT
Text Area	×	×	×
Advanced Text Area	×	×	
Tree	×	×	
Menu	×	×	
Tab Folder	×	×	
Toolbar	×	×	×
Spinner	×	×	
Spinner	×	×	
Table	×	×	×
Advanced Table	×	×	

之所以 SWT 看起来要比 Swing 容易开始得多，是因为 SWT 做了很多在 Swing 中需要开发人员去做的事，例如 Model-View-Controller 模式的应用，可插拔的 look and feel 机制等，但是 SWT 的这些优点使得它存在着一个潜在的问题：资源释放。Swing 与 AWT 遵循 Java 规范中的资源自动释放（automatic resources disposal）原则，因此它们不必考虑这个问题，它们可以将释放资源这个工作交给 JRE 的垃圾回收线程去做。而 SWT 在不再使用本地资源时，需要开发人员在程序代码中显式地释放资源。

从结构上来说，Java AWT 的构件库使用 Java 与 C 的类库来共同实现跨平台的特性。由于用户接口是 SUN 公司实现的，所以用户接口不会直接调用操作系统的窗口部件，它需要和 Java 语言一样，实现平台无关性。AWT 的类库还不完善，缺少一些常用的构件，并且缺乏灵活性，所以 Swing 基于 AWT 开发时着重解决的就是使用上的灵活性问题，并且补充了很多有用的构件。图 1.5 比较清楚地阐述了 SWT 与 Swing 在实现机制上的不同。

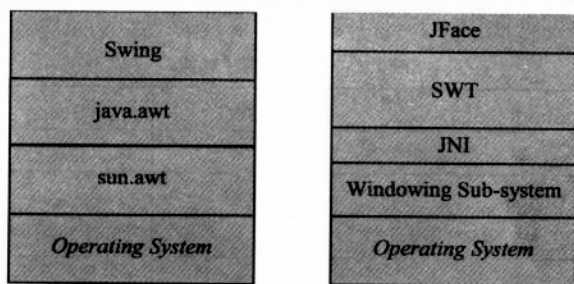


图 1.5 SWT 与 Swing 的结构比较

1.5.2 SWT 的优势和不足

综合以上讨论可以发现，SWT 的长处在于：

- ❑ Look 和 Feel 与本地操作系统对应。
- ❑ 简单实用的 API 可以使开发人员快速上手。
- ❑ 由于本地的 JNI 调用机制，SWT 应用程序运行速度非常迅速。
- ❑ 可以仿造本地操作系统的风格画出本地操作系统中没有的部件。

SWT 的不足在于：

- ❑ 每一种操作系统都需要有相匹配的 JNI 程序供 SWT 调用。
- ❑ 没有 Swing 那样灵活。

总的来说，如果需要利用 Java 语言面向对象、跨平台等种种优势，同时又希望创建一套和本地操作系统风格兼容的 GUI 应用，那么 SWT 应该是首选的。

1.6 SWT 与 JFace、Eclipse

SWT 与 JFace、Eclipse 关系可以用以下两条关键字来概述：

- ❑ JFace 是 SWT 的扩展。
- ❑ Eclipse 的 UI 界面基于 JFace。

图 1.6 清晰地描述了 SWT 与 JFace、Eclipse 的关系。

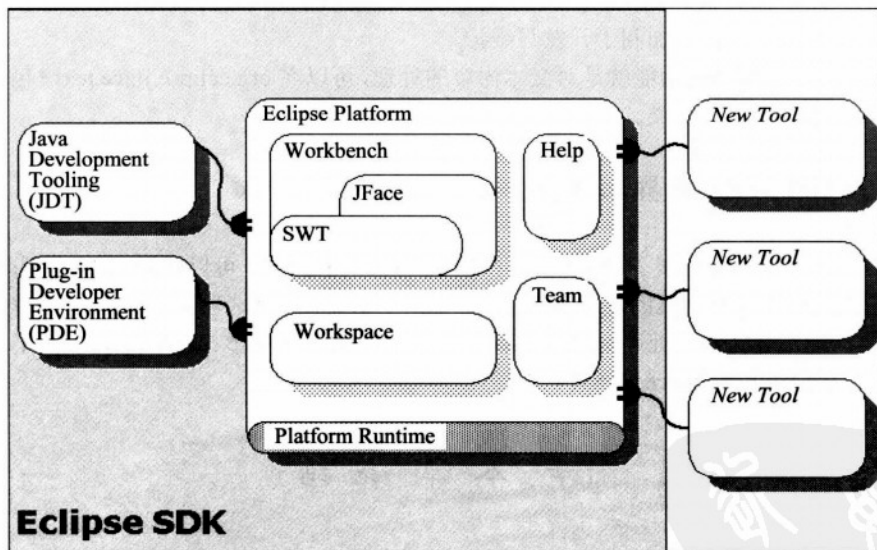


图 1.6 Eclipse 体系构架

1.6.1 JFace 是 SWT 的扩展

JFace 与 SWT 的关系好比 Microsoft 的 MFC 与 SDK 的关系。JFace 是基于 SWT 开发的，其 API 比 SWT 更加易于使用，但功能却没有 SWT 来的直接。例如，下面的代码应用 JFace

中的 `MessageDialog` 打开一个警告对话框：

```
MessageDialog.openWarning(parent, "警告", "警告消息");
```

如果只用 SWT 完成以上功能，语句不会少于 30 行！

JFace 原本是为更加方便地使用 SWT 而编写的一组 API，其主要目的是为了开发 Eclipse IDE 环境，而不是为了应用到其他的独立应用程序。因此在 Eclipse 2.1 版本之前，很难将 JFace API 完整地从小核 API 中剥离出来，总是要多多少少地导入一些非 JFace 以外的 Eclipse 核心代码类或接口才能得到一个没有任何编译错误的 JFace 开发包。但目前 Eclipse 组织似乎已经逐渐意识到了 JFace 在开发独立应用程序时起到的重要作用。在 Eclipse 当前的版本中，JFace 已经变成了和 SWT 一样的完整独立的开发包。

对开发人员来说，在开发一个图形构件时，比较好的方式是先到 JFace 包去找一找，看是不是有更简洁的实现方法，如果没有再用 SWT 包去自己实现。例如，JFace 为对话框提供了很好的支持，除了各种类型的对话框（例如上面用的 `MessageDialog`，或是带有标题栏的对话框）。如要实现一个自定义的对话框也最好从 JFace 中的 `Dialog` 类继承，而不是从 SWT 中的对话框中继承。

应用 JFace 中的首选项包（`Preference`）中的类很容易为自己的软件做出一个很专业的配置对话框。对于 `Tree`、`Table` 等图形构件，它们在显示的同时也要和数据关联。例如，`Table` 中显示的数据，在 JFace 中的 `View` 包中为此类构件提供了 MVC 方式的编程方法，这种方法使显示与数据分开，更加利于开发与维护。

JFace 中提供最多的功能就是对文本内容的处理，可以在 `org.eclipse.jface.text.*` 包中找到数十个与文本处理相关的类。

1.6.2 Eclipse 的 UI 界面基于 JFace

Eclipse 的 UI 界面扩展了 JFace。在 Eclipse 的工作平台中，可以找到 JFace 的身影，例如，首选项的设置和对话框的使用等，这些都是基于 JFace 的首选项和对话框组件。对 JFace 中很重要的一部分文本的处理组件，可以在 JDT 中发现它的身影。Eclipse 是一个平台，它的 UI 界面部分是基于 JFace 和 SWT 的。

1.7 本章小结

本章首先回顾了 Java 桌面应用的历史，然后阐述了 SWT 诞生的背景和 SWT 的优势所在。最后描述了 SWT 与 JFace 和 Eclipse 的关系。通过本章的学习，读者对 SWT 有了一个大致了解。

第2章 配置 SWT 开发环境

本章首先将针对两种配置开发环境的方式进行详细的说明，然后讲述如何下载和安装 Eclipse 官方提供的 Visual Editor (VE) 插件，该插件专门针对图形界面的开发，可以极大地减少代码开发量。最后编写一个简单的 SWT 程序，使读者对 SWT 程序有感性的认识。

2.1 下载和安装 Eclipse

在开始进行 SWT 应用程序开发之前，需要搭建好 SWT 应用程序的运行环境。需要说明的是，SWT 应用程序不仅可以在 Eclipse 平台中以插件的形式使用，也可以将其脱离出来，像使用一般的 Java 类包一样在 Eclipse 平台之外单独使用。获取 SWT 开发工具包（也就是 jar 文件）可以有两种方式：

- ❑ 可以在 Eclipse 官方网站下载 Eclipse SDK 的任意一个版本之后，从 Eclipse SDK 的插件目录，也就是解压缩文件后的 Plug-in 目录中找到相应的 SWT 支持插件，从这些插件中取出开发支持包（在 Windows 系列中是 jar 形式）和基于本地操作系统的动态链接库（在 Windows 系列中是.dll 形式）以运行 SWT 应用程序。
- ❑ 直接在 Eclipse 官方网站上下载最新的 SWT 开发工具包，此方法适用于不使用 Eclipse 作为开发工具，而又想进行 SWT 开发的用户。

2.1.1 Eclipse 下载页面介绍

使用浏览器打开 <http://www.eclipse.org/downloads/> 页面，进入到 Eclipse 官方下载界面，该界面如图 2.1 所示。

笔者在写该书时，最新发布的版本是 Eclipse SDK 3.1.2，但是这里并不使用该版本，而选择使用 Eclipse SDK 3.1 版本。其原因有以下几点：

- ❑ Eclipse SDK 3.1 已经是一个相对完善和稳定的版本，如果作为开发用，当然会选择一个比较稳定的版本。
- ❑ 到笔者写该书时，Eclipse 的语言包只支持 Eclipse SDK 3.1.1，如果使用 Eclipse SDK 3.1.2 则会有一部分不能够汉化。而语言包是可以向下兼容的，也就是说如果安装 Eclipse SDK 3.1.1 语言包，对于该版本以下的 Eclipse 是兼容的。
- ❑ 要下载使用的 Visual Editor 也存在 Eclipse 版本兼容的问题，所以这里选择安装 Eclipse SDK 3.1 版本。

 注意：本书中所使用的 Eclipse 都为 Eclipse SDK 3.1 版本。



图 2.1 Eclipse 官方下载页面

2.1.2 下载 Eclipse

(1) 单击图 2.1 所示的 All versions 链接进入到 Eclipse 其他版本的下载界面，或者直接访问 <http://download.eclipse.org/eclipse/downloads/index.php> 进入到该页面。该网页的界面如图 2.2 所示。

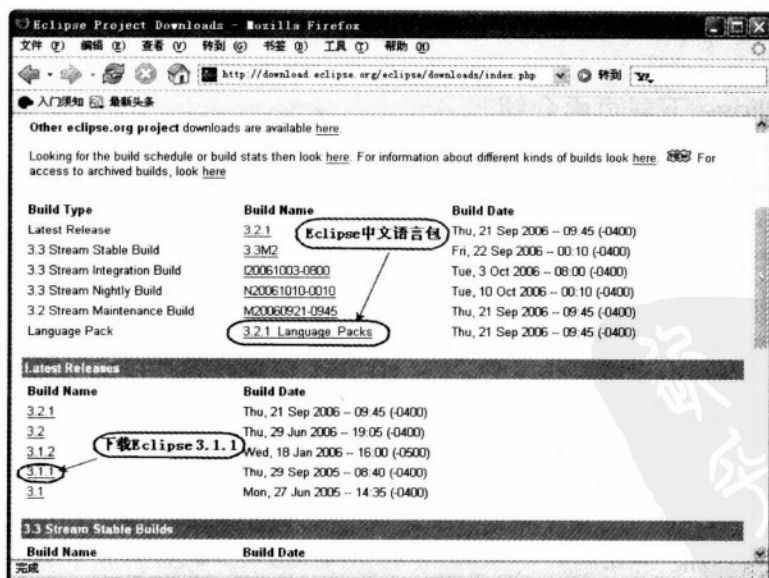


图 2.2 下载 Eclipse 的界面

(2) 在该页面上, 可以看到 Eclipse 的不同版本, 选择 Eclipse 3.1 版本进行下载 (或者直接访问 <http://download.eclipse.org/eclipse/downloads/drops/R-3.1-200506271435/index.php>)。稍后将会讲述如何安装语言包。

Eclipse 发布的不同版本介绍:

- ❑ Releases 版本: 发布版本。一般是最稳定和最完善的版本, 发布版本的版本号通常会以字母 R 开头, 如 R1.0, R2.0。如果不是发布版本, 通常是以发布当日的日期命名的, 如软件版本 20011027, 意思是软件的发布日期是 2001.10.27。
- ❑ Stable Builds 版本: 稳定版本。一般推出新功能后, 经过一定时间的测试, 基本上稳定的版本。
- ❑ Nightly Builds 版本: 每日发布版本。每日发布版本并没有进行充分测试。这个版本是测试版本。

说明: Releases 版本是最可靠的版本。如果喜欢尝试新的功能可以使用 Stable Builds 版本。

(3) 在该下载页面上, 用户可根据不同的操作系统进行下载。例如, Windows 用户可以单击如图 2.3 所示的链接后选择镜像进行下载。

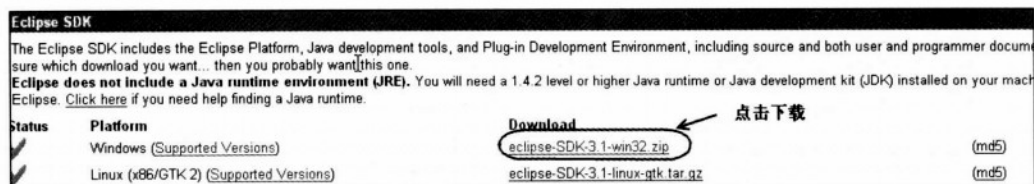


图 2.3 下载 Eclipse SDK

(4) 下载后的文件名为 eclipse-SDK-3.1-win32.zip, 然后解压缩文件。例如这里将该压缩文件解压缩到 F:\javaDev 文件夹下。解压缩后的 F:\javaDev\eclipse 文件目录结构如图 2.4 所示。



图 2.4 Eclipse 的文件目录结构

(5) 这里有两个很重要的文件夹 features 和 plugins, 一般安装插件时, 会覆盖这两个文件夹 (这里所说的覆盖并不是删除原来的文件, 而是按照这种目录结构将新的文件复制到文件夹下)。features 文件夹中存放的是有关 Eclipse 功能描述的信息。Plugins 文件夹中

存放的是 Eclipse 运行所用的各种插件包。

运行文件目录中的 eclipse.exe 文件, 如果出现 Eclipse 的工作界面则表示 Eclipse 已经安装成功了。

注意: 如果此时双击 eclipse.exe 文件后不能运行, 则要检查是否已安装 Java 虚拟机, 并且保证虚拟机的版本与 Eclipse 的运行环境相一致。例如这里, Eclipse SDK 3.1 版本所需要的虚拟机的版本为 1.4.2 以上版本。

2.1.3 安装 Eclipse 语言包

Eclipse 的工作平台是英文的, 可以安装官方提供的多语言的插件来配置成多种语言。首先单击图 2.1 所示的 Eclipse3.1.1 语言包链接(也可以直接访问地址 http://download.eclipse.org/eclipse/downloads/drops/L-3.1.1_Language_Packs-200510051300/index.php)。

在该页面上, 可以根据不同的操作系统选择所需要下载的语言包。例如, Windows 用户可以选择图 2.5 所示的语言包进行下载。

SDK Language Packs	Windows 98/ME/2000/XP
NLpack1 - German, Spanish, French, Italian, Japanese, Korean, Portuguese (Brazil), Traditional Chinese and Simplified Chinese.	NLpack1 FeatureOverlay-eclipse-SDK-3.1.1.zip NLpack1-eclipse-SDK-3.1.1a-win32.zip
NLpack2 - Czech, Hungarian, Polish and Russian	NLpack2 FeatureOverlay-eclipse-SDK-3.1.1.zip NLpack2-eclipse-SDK-3.1.1a-win32.zip
NLpackBidi - Arabic	NLpackBidi FeatureOverlay-eclipse-SDK-3.1.1.zip NLpackBidi-eclipse-SDK-3.1.1a-win32.zip

图 2.5 语言包下载界面

下面来看一下如何一步步地安装 Eclipse 的语言包插件。

(1) 首先关闭 Eclipse, 确保已经退出 Eclipse 工作区。

(2) 解压缩下载的两个文件到 Eclipse 的目录中。如图 2.6 所示, 在语言包文件上右击, 在弹出的快捷菜单上选择“解压到当前文件夹”命令。其实, 打开压缩文件可以看到压缩文件中也有 features 和 plugins 两个文件夹, 只需要按照 Eclipse 的目录结构将文件复制过去即可。

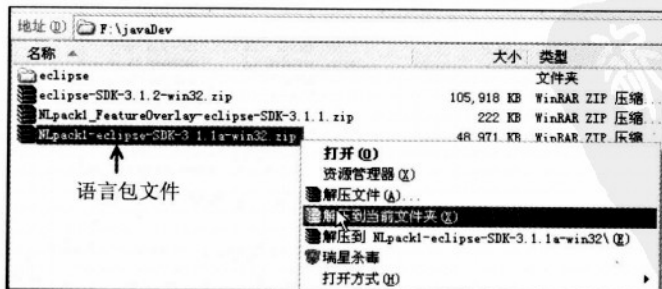



图 2.6 解压缩语言包

 **小知识:** Eclipse 的插件都是以这种形式安装的, 只需覆盖 features 和 plugins 文件夹即可。

(3) 重新启动 Eclipse, 便可以看到安装中文版后的 Eclipse 工作界面了。汉化后的 Eclipse 工作界面如图 2.7 所示。

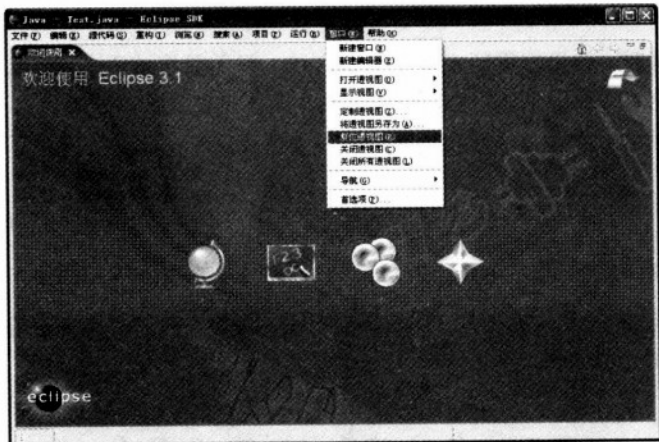



图 2.7 Eclipse 的中文环境

 **注意:** 如果此时启动后还有部分没有汉化, 这时需要反复启动 Eclipse 即可。

2.1.4 在不同的语言中切换

Eclipse 的语言包是以插件形式存在的, 它的管理也与其他插件一样, 即需要时可以启用, 不需要时只需禁用该插件即可。

(1) 选择“帮助”|“软件更新”|“管理配置”命令, 出现如图 2.8 所示的界面。若在英文工作环境下, 则为 Help|Software Updates|Manage Configuration 命令。

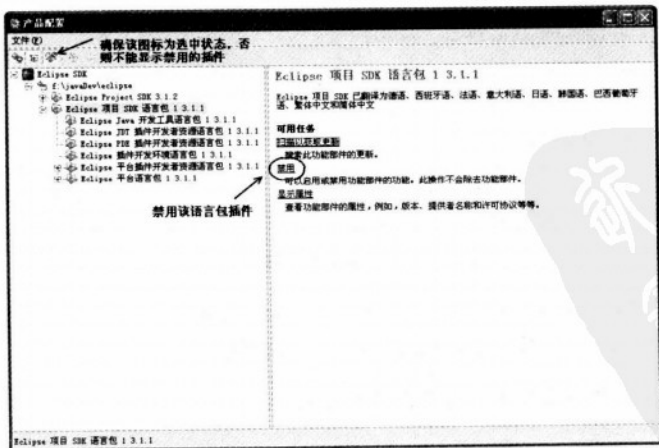



图 2.8 插件管理界面

(2) 在工作区的左侧是所有已经安装的插件，右侧是插件的属性。图中“Eclipse 项目 SDK 语言包 1 3.1.1”是以插件形式存在的。展开菜单，在右侧的属性栏中可以看到一个“禁用”选项（英文环境下显示为 Disable）。单击即可卸载该语言包插件。

 **注意：**更改此设置后 Eclipse 会关闭，然后重新启动。

(3) 同样道理，如果想重新安装中文语言包，只需要将该语言包的插件状态设置为可用状态即可。如图 2.9 和图 2.10 所示为插件禁用状态和启用状态的界面。



图 2.9 卸载插件后的显示界面

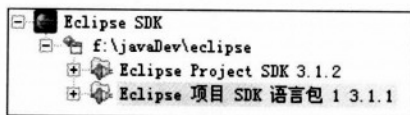



图 2.10 启用插件后的显示界面

 **说明：**语言包插件的安装也适用于其他的 Eclipse 插件，如果想要查看启用或禁用某个插件，操作方法与此类似。

2.2 直接获取 SWT 工具包

还有另一种获得 SWT 工具包的方法，可以到 Eclipse 的官方主页上直接下载。“<http://www.eclipse.org/swt/>”是 SWT 项目的官方主页，在这里可以找到有关 SWT 的最近消息，以及版本的介绍等。在这里，可以下载到最新的 SWT 版本。

其实也可以在 Eclipse 下载页面中选择下载 SWT 开发包。例如 Eclipse 3.1 的下载地址为 <http://download.eclipse.org/eclipse/downloads/drops/R-3.1-200506271435/index.php>。然后在页面的底部可以看到如图 2.11 所示的下载界面。

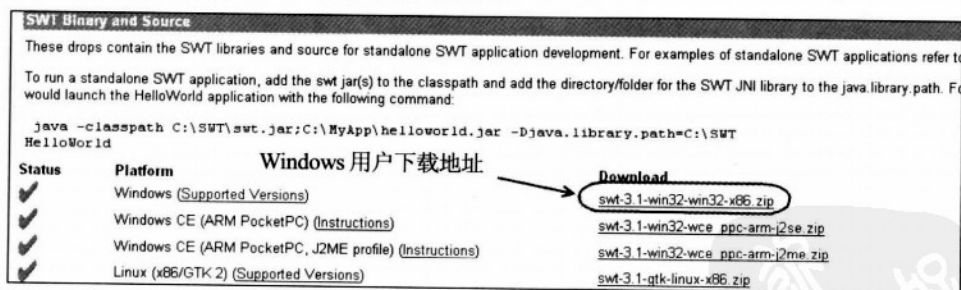



图 2.11 SWT 下载界面

下载图示中的地址文件后，并设置此 jar 包的 classpath 就可以运行 SWT 程序了。这种获取 SWT 开发包的方式适用于不使用 Eclipse 开发的用户。

 **注意：**Eclipse 3.1 中所自带的 SWT 版本为 SWT 3.1.0。

2.3 下载和安装 Visual Editor

Visual Editor 是一个开发 GUI 界面的工具，它能使开发 Swing/JFC 和 SWT/RCP 界面的工作变得轻松。它可以通过拖动的方式创建所需要的控件。虽然使用它可以大大简化程序的开发量，但并不能取代手工去输入代码。这里所要强调的是：这个插件是可选的，也可以不使用该工具开发 SWT 程序。

2.3.1 Visual Editor 的下载

Visual Editor 项目的官方主页是 <http://www.eclipse.org/vep/WebContent/main.php>，读者可以访问这里了解 Visual Editor 的最新动向。这里只给出 Visual Editor 的下载地址：<http://download.eclipse.org/tools/ve/downloads/drops/R-1.1-200507221721/index.html>，打开该页面后下载如图 2.12 所示的文件。

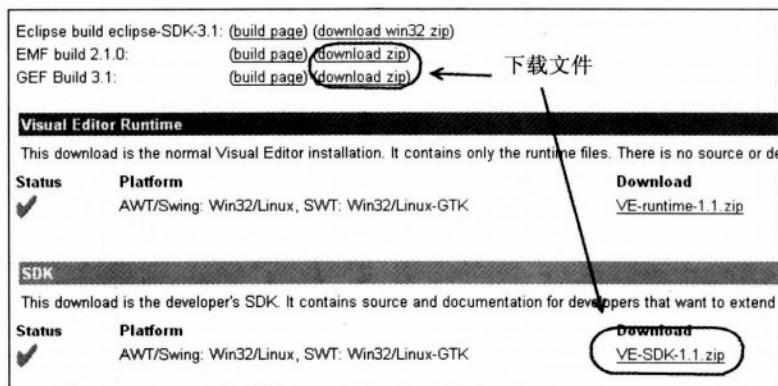


图 2.12 Visual Editor 的下载界面

在下载 Visual Editor 时同时也要下载 EMF 2.1.0 和 GEF 3.1，这两个插件是运行 Visual Editor 所必需的，否则不能运行。

⚠注意：这里下载的是 Visual Editor 1.1 的版本，该版本运行的环境是 EMF 2.1.0 和 GEF 3.1，要同时下载这 3 个插件。

💡小知识：SDK 和 runtime 的区别。一般在下载插件时都可以选择 SDK 或者是 runtime 版本。例如下载 Visual Editor 时，可以下载 VE-runtime-1.1.zip 或 VE-SDK-1.1.zip。runtime 版本只包括运行该插件的.class 文件和其他的一些配置文件。但 SDK 文件不仅包括 runtime 文件，还包括.java 的源文件，作为学习研究之用，最好下载 SDK 版，这样可以方便查看源代码，有利于学习。

2.3.2 Visual Editor 的安装

Visual Editor 既然是 Eclipse 的插件，也就可以按照安装插件的方法来安装，读者可以参考 Eclipse 语言包的安装方法。

(1) 将下载下来的文件复制到 F:\javaDev 文件夹下，下载的 3 个文件如图 2.13 所示。



图 2.13 Visual Editor 下载文件

(2) 按照同样的方法，首先关闭 Eclipse，然后分别将 3 个文件解压缩，最后重新启动 Eclipse。安装完 Visual Editor 的界面效果如图 2.14 所示。

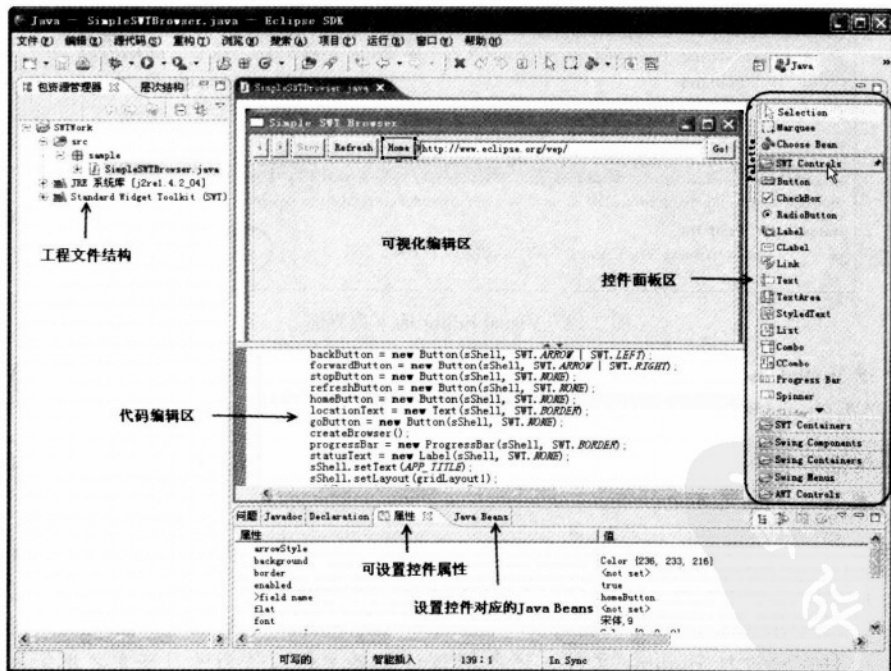



图 2.14 Visual Editor 的工作界面

(3) 工作区的右侧是控件面板，用户可以选中所需控件，直接添加到可视化工作区中，系统会自动生成该控件的代码，大大简化了工作量。但是，并不能完全依靠编辑器，大多数情况下需要自己手动输入代码。

 **注意：**虽然在这里介绍了可视化的编辑器，但并不赞成初学者过多使用编辑器。对于初学者来说，不要害怕麻烦，要从基础做起，打好基本功才是最重要的。

2.4 第一个 SWT 程序

2.4.1 创建 SWT 程序

现在，一切开发 SWT 的环境已经配置好了，接下来要看一下如何创建 SWT 应用程序。要创建 SWT 程序，首先要建立一个 Eclipse 的项目工程文件。建立项目文件的步骤如下：

(1) 打开 Eclipse，选择“文件”|“新建”|“项目”命令，在弹出的“新建项目”对话框中选择“Java 项目”，单击“下一步”按钮，弹出如图 2.15 所示的对话框。

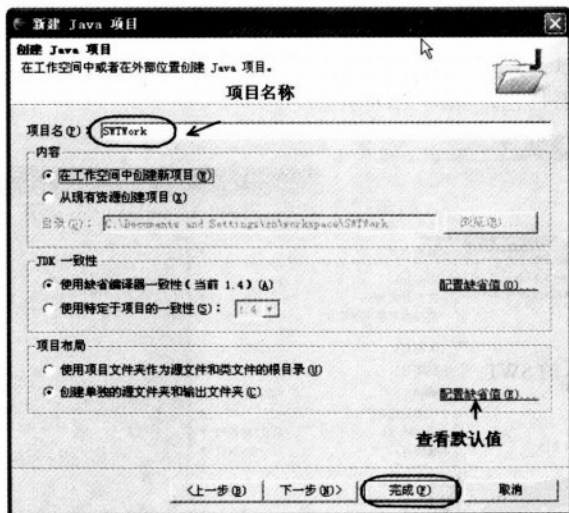


图 2.15 “新建 Java 项目”对话框

(2) 输入项目名称后，单击“完成”按钮，即新建了一个名为 SWTWork 的项目。

(3) 在项目工程下，选择“文件”|“新建”|“其他”命令，在弹出的“新建”对话框中选择 Simple SWT Browser，单击“下一步”按钮，如图 2.16 所示。

(4) 弹出如图 2.17 所示的对话框，输入包名“firstSWT”后，单击“完成”按钮，即完成了一个 Visual Editor 自带的一个 SWT 程序的小例子。

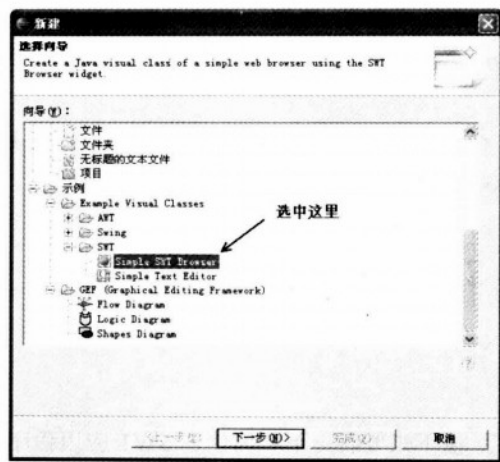


图 2.16 “新建”对话框

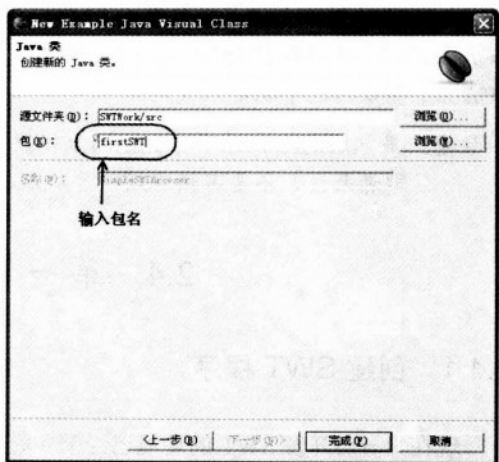


图 2.17 输入包名

2.4.2 编译和运行程序

创建完 SWT 项目后，会发现在导航栏中自动添加了 Standard Widget Toolkit 包，这个即是 SWT 所使用的工具包，如图 2.18 所示。



图 2.18 运行 SWT 程序示意图

要运行 SWT 程序很简单，只要右击类文件，在弹出的快捷键菜单中选择“运行方

式”|“SWT 应用程序”命令即可，如图 2.18 所示。本例创建的是 Visual Editor 自带的一个浏览器 SWT 程序。程序运行后效果如图 2.19 所示。

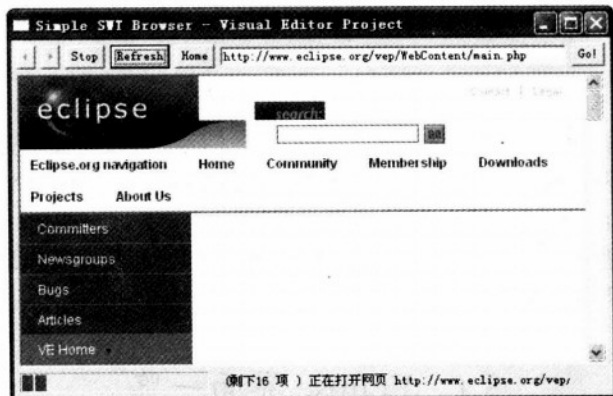


图 2.19 SWT 的浏览器

到此为止，总算是看到了一个真正的 SWT 程序。这仅仅是开始，随着内容的深入，读者会领略到更多有关 SWT 的知识。

2.5 本章小结

通过本章的学习，读者应该学会如何在 Eclipse 的官方网站上来获取有用的信息，并且对 Eclipse 的一些术语有所了解。读者还应学会如何来安装和卸载 Eclipse 的插件。最后读者要明确一个概念，通过下载 Eclipse 来获取 SWT 开发包并不是唯一的途径，也可以选择单独下载。最后，一个 SWT 浏览器的程序使读者对 SWT 程序有了一个感性的认识，通过本书的学习，必将带你进入一个精彩的 SWT 世界。



第 3 章 Eclipse 开发环境概述

本章对 Eclipse 的开发环境作简单的介绍。如果读者已经对 Eclipse 的开发环境非常熟悉，那么本章可以作为补充知识；如果读者还不曾用过 Eclipse，那么本章将会使读者快速掌握 Eclipse 的开发环境，扫清后文学习的障碍。

本章讲述的内容主要是使用 Eclipse 开发时常用的技巧，相信读者如果掌握了这些技巧，必定大大提高代码开发的效率。

3.1 Eclipse 界面一览

作为一种 IDE 开发工具，Eclipse 有自己独特的开发环境界面。下面就来具体认识一下 Eclipse 界面的组成部分。如图 3.1 所示为 Eclipse 的开发环境界面。

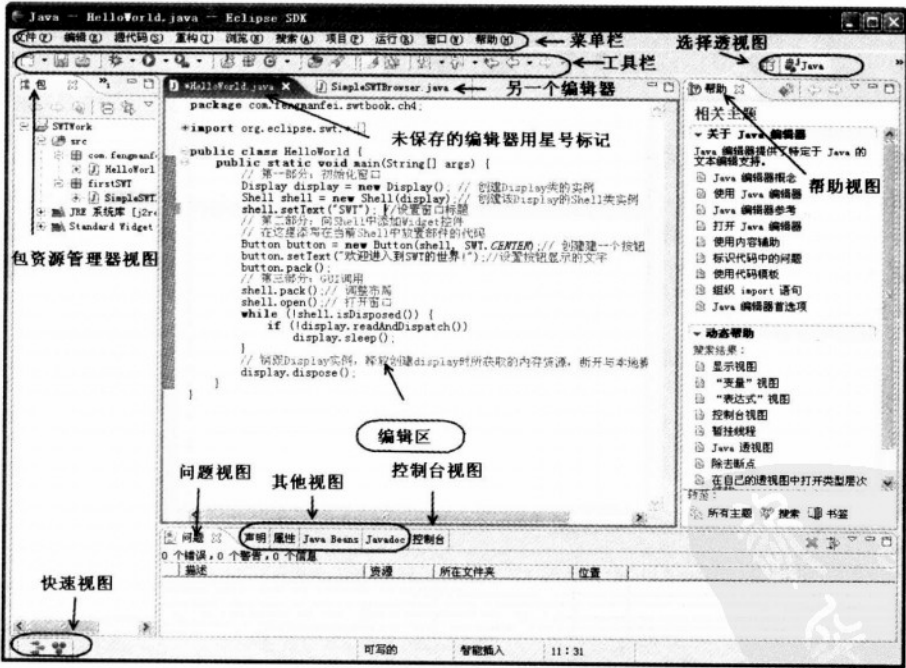


图 3.1 Eclipse 主窗口界面

如图 3.1 所示，菜单栏和工具栏是程序中常见的组件，这里不再过多介绍。要想理解 Eclipse 工作区，必须要掌握 3 个概念：编辑器、视图和透视图。

- ❑ 编辑器 (Editor)：编辑器是用户打开文件后进行编辑的窗口。一个工作区内可以同时打开多个编辑器以对不同文件进行编辑。未保存的编辑器会在选项卡的标签上以星号标记。编辑器也可以放大或缩小。当同时打开多个编辑器时，可以按拖动的方式设置编辑器的布局，使之更适合自己的喜好。总之，编辑器是编写程序代码的地方，是工作区最重要的部分。
- ❑ 视图 (View)：视图是方便浏览工作区信息的导航区。如图 3.1 所示有“包资源管理器视图”，可以按包结构来浏览 Java 源文件。有“帮助视图”，可以直接获取帮助。有“问题视图”显示问题列表。Eclipse 提供了各种各样的视图，可以通过选择“窗口”|“显示视图”|“其他”命令来选择想要浏览的视图。总之，视图是方便浏览信息的地方，可以使用户更加快捷地进行开发。
- ❑ 透视图 (Perspective)：透视图的作用是管理视图和工作区布局。用户可以按照自己的喜好，将视图的布局保存为一个透视图，这样，一旦选择透视图后，就会按照设定的透视图来显示视图布局。Eclipse 内置了一些常用的透视图，可以通过选择“窗口”|“打开透视图”|“其他”命令来选择不同的透视图。另外，如果要想保存当前视图布局，可以通过选择“窗口”|“将透视图另存为”命令来保存。

编辑器、视图和透视图的关系如图 3.2 所示。

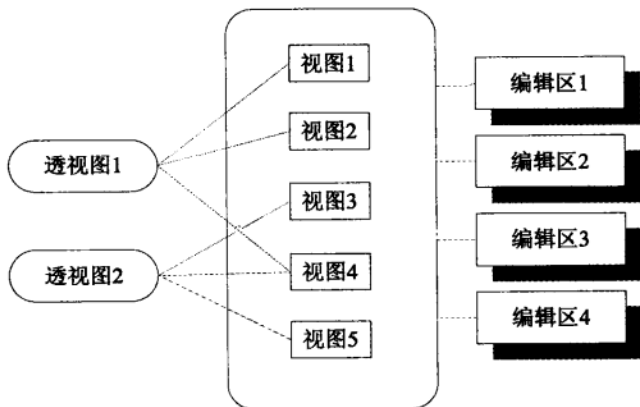


图 3.2 编辑器、视图和透视图的关系图

3.2 Eclipse 项目的文件结构

认识了 Eclipse 的工作环境，下面就来看一下如何开发 Java 程序。包括如何创建类、如何编译类文件、如何导入编译时的类包、如何运行程序等。

3.2.1 设置编译后.class 文件的保存目录

在第 2 章中已经介绍了如何创建一个项目工程。下面介绍在项目工程中创建 Java 文件

的过程。

(1) 在项目工程下，选择“文件”|“新建”|“类”命令，在弹出的“新建 Java 类”对话框中选择输入包名和类名，单击“完成”按钮，如图 3.3 所示。

(2) 创建类后，该类文件将会显示在“包资源管理器”视图中，如图 3.4 所示。此时可以清楚地看到该类所在的包和类名。

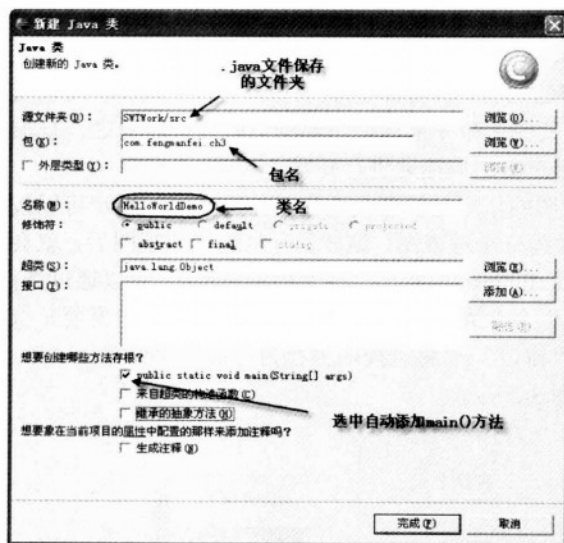


图 3.3 “新建 Java 类”对话框



图 3.4 类结构

那么现在读者可能会有疑问了，这里显示的只是 Java 的源文件，那么编译以后的.class 文件会保存到哪里呢？能不能指定输入后的.class 文件的存放位置呢？

(3) 在“包资源管理器”中的项目名上右击，在弹出的快捷菜单中选择“构建路径”|“配置构建路径”命令，弹出如图 3.5 所示的“SWTWork 的属性”对话框。也可以通过选择“项目”|“属性”|“Java 构建路径”|“源代码”命令来显示该对话框。

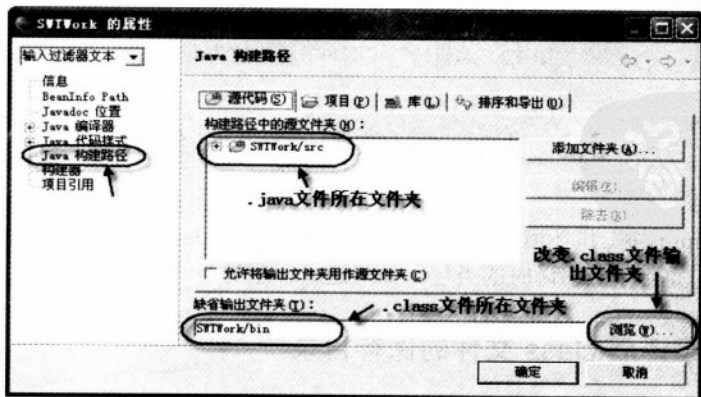


图 3.5 “SWTWork 的属性”对话框 (1)

(4) 从图 3.5 中可以看出, .class 文件所在的文件夹是“SWTWork/bin”, 那么所有编译后的.class 文件都保存在 bin 这个文件夹下。例如, 在这里 SWTWork 项目的保存路径为“E:\Documents and Settings\jan\workspace\SWTWork”, 那么在“E:\Documents and Settings\jan\workspace\SWTWork\bin”文件下就可以找到这些编译后的类文件了。

(5) 若要改变.class 文件所在的文件夹, 可以单击“浏览”按钮, 选择所要输出的文件夹即可。同样, 也可以改变.java 源文件所存放的目录。

3.2.2 导入项目使用的包

Eclipse 生成的项目文件会自动将 JRE 系统库包含进来。如果在该项目中使用了其他的 Jar 包的类, 则需要先导入到项目中。

单击如图 3.5 所示的“库”标签, 或者选择“项目”|“属性”|“Java 构建路径”|“库”命令, 打开如图 3.6 所示的对话框。

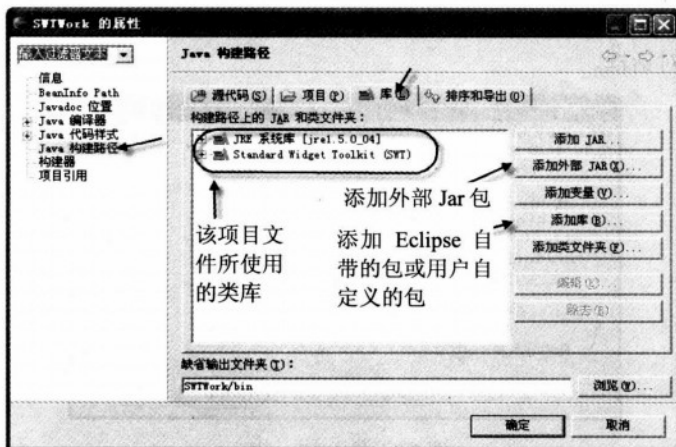


图 3.6 “SWTWork 的属性”对话框 (2)

这里显示了该项目所使用的类库, 例如这里已经包含了 JRE 系统库和 SWT 类库。导入类包通常有两种情况: 一种是使用第三方提供的 Jar 包, 比如常见的数据库连接的类包; 一种是 Eclipse 自带的类包。当然还可以导入为打包成 Jar 包的.class 文件夹。这里只介绍通用的两种导入包的方式。

1. 导入第三方的类包

例如, 存在一个 E:\tools\mysql.jar 包文件 (该包为连接 MySQL 数据库包), 此时单击如图 3.6 所示的“添加外部 JAR”按钮, 在弹出的对话框中选择此文件后, 导入后的包如图 3.7 所示。

若此时有该包所对应的源代码文件和 JavaDoc 文件, 也可以关联进来以方便参考。将包添加到项目工程中后, 在“包资源管理器”视图中会自动显示出该包中所有的类, 如图 3.8 所示。

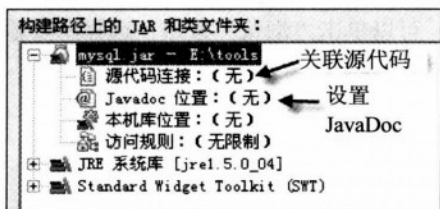


图 3.7 导入后的包



图 3.8 添加包后的“包资源管理器”视图

2. 导入 Eclipse 自带的包

例如，此时要导入 JFace 的包到项目中，此时单击如图 3.6 所示的“添加库”按钮，在弹出的对话框中选择 Standard Widget Toolkit (SWT)选项，然后单击“下一步”按钮，弹出如图 3.9 所示的对话框。

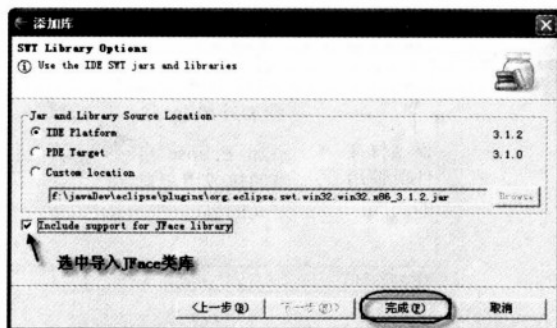


图 3.9 导入 JFace 库

单击“完成”按钮，即可将 SWT 包和 JFace 包导入工程中。

3.2.3 设置编译方式

学习了如何新建 Java 类文件，也了解了将所需要的包导入到项目中来，下一步就要了解如何将.java 文件编译成.class 文件了。Eclipse 提供了一种很方便的编译文件的方式，即“自动构建”。也就是说，在保存.java 源文件时，直接将.java 文件编译成.class 文件，而不需要再进行编译。

(1) 若查看该功能是否启用，单击“项目”菜单，确保“自动构建”菜单项为选中状态，如图 3.10 所示。

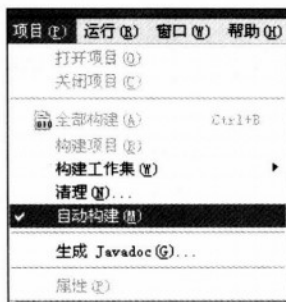


图 3.10 “项目”菜单

(2) 若此时未选中“自动构建”，那么就需要手动进行编译文件了。用户可以选择“全部构建”命令来编译所有的源文件（也可以使用 Ctrl+B 快捷键）。也可以选择“清理”命令将删除掉已经编译的.class 文件。

3.2.4 运行程序

在第 2 章中虽然只介绍了最简单的运行程序的方式，但也可以对运行程序参数进行更多的设置。在所要运行的文件上右击，在弹出的快捷菜单中选择“运行方式”|“运行”命令，弹出如图 3.11 所示的“运行”对话框。

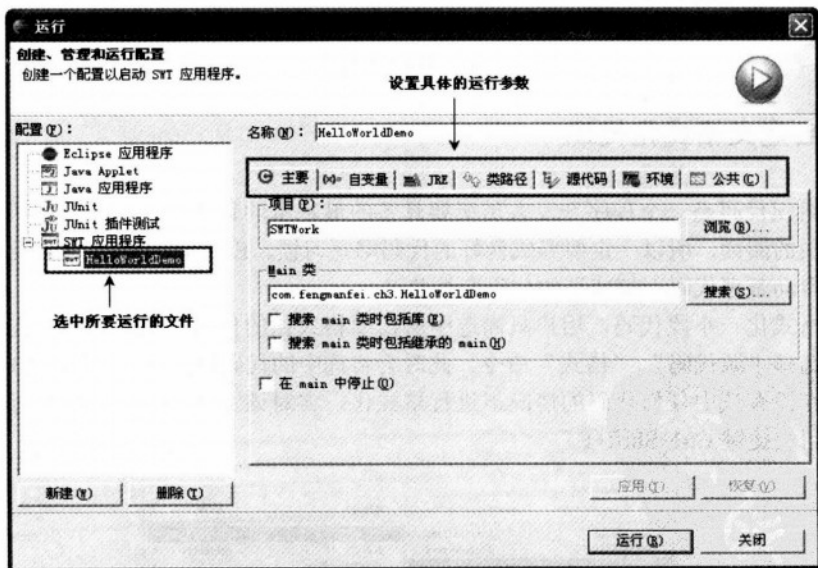


图 3.11 设置运行参数示意图

在此界面可以为运行程序设置各种运行参数，包括自变量、类库、类路径、环境变量等。

3.3 常用的代码编辑功能

使用 Eclipse 进行 Java 开发时，可以使用 Eclipse 的许多辅助功能，例如格式化代码、代码的重构、代码比较等，这些功能可以大大提高开发的效率。

3.3.1 添加注释

Eclipse 提供了方便添加代码注释的方法，用户只需在代码处输入 “/**” 之后按 Enter 键，就自动完成了注释的标记。

同时，为了生成 JavaDoc 文件，通常会在注释中添加代码标记。这对 Eclipse 来说也非常容易，只需要输入 “@” 标记，Eclipse 会自动提示，和编辑 Java 代码的提示功能一样，如图 3.12 所示。

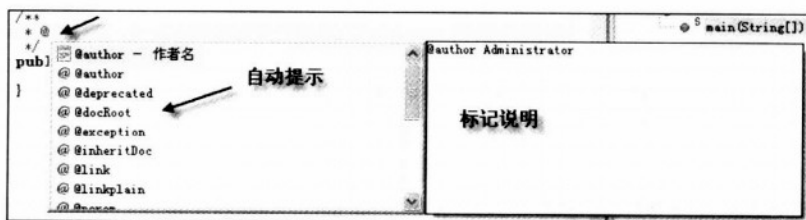


图 3.12 注释自动提示

3.3.2 自定义格式化代码

良好组织代码是一个程序开发人员所要具备的最基本的素质，格式化的代码有益于其他开发人员的阅读，所以一定要养成良好的代码书写习惯。Eclipse 提供了格式化代码的功能，能够自动整理代码，使凌乱的代码变得整洁。

要想格式化一小段代码，用户只需选中所需要格式化的代码，然后右击，在弹出的快捷菜单中选择“源代码”|“格式”命令，此时会将选中的代码格式化，如图 3.13 所示。

如果在没有选中任何代码的情况下进行格式化，将对该文件的所有代码进行格式化。也可以使用快捷键 Ctrl+Shift+F。

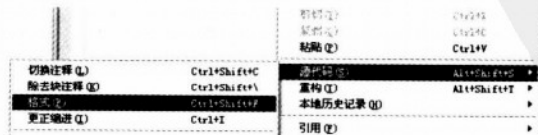


图 3.13 格式化代码菜单

那么，格式化以后的代码是按照什么格式进行格式化的呢？怎么才能按照用户设定的

方式进行格式化代码呢？选择“窗口”|“首选项”命令，在弹出的“首选项”对话框中选择“格式化程序”，如图 3.14 所示。

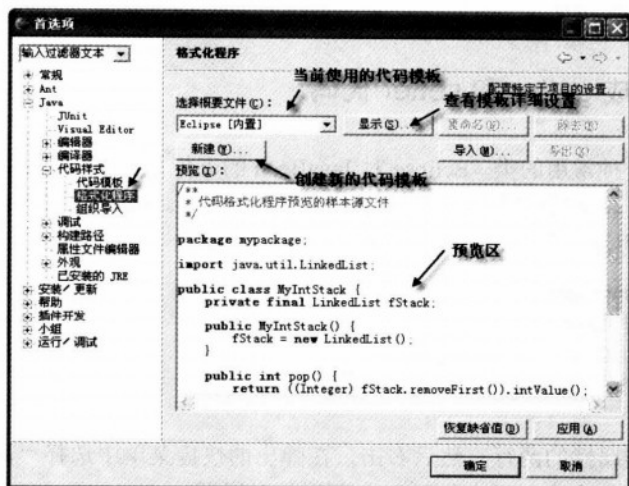


图 3.14 “首选项”对话框

Eclipse 内置了“Eclipse”、“Java 约定”和“Eclipse 2.1”3 个格式化代码的模板，这 3 个模板是不可以改变的。用户如果要自定义代码模板，需要在这 3 个内置的模板上进行修改。自定义代码模板的步骤如下：

(1) 单击如图 3.14 所示的“新建”按钮。在弹出的对话框中输入模板的名称并选择格式化设置文件。例如，模板的名称为 myStyle，并选择 Eclipse，单击“确定”按钮，弹出如图 3.15 所示的对话框。

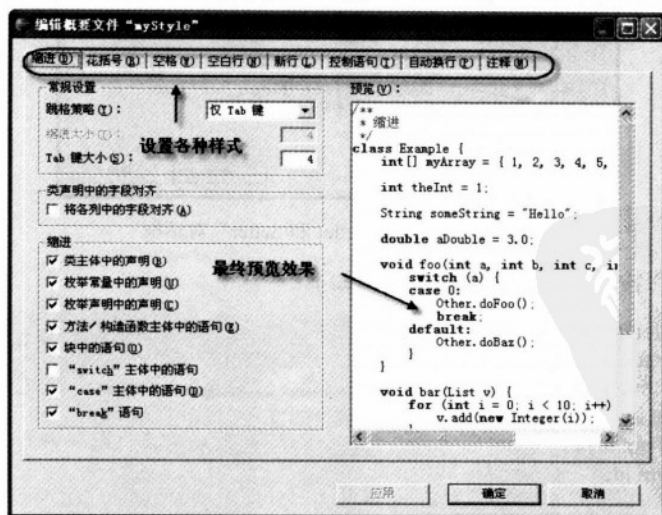


图 3.15 “编辑概要文件 ‘myStyle’ ”对话框

(2) 设计代码想要的样式, 这里就不多作介绍了。可以设置的选项很多, 可以依照不同的喜好来进行设置。这样, 如果以后再进行格式化代码的操作, 就会按照刚才定义的代码格式进行格式化代码了。

3.3.3 自动生成 getter 和 setter 代码

JavaBean 是一种常用的类, Eclipse 对 JavaBean 的支持很好用。JavaBean 类中有类的属性, 也有设置和获取类属性的方法——getter 和 setter 方法。Eclipse 可以对类的属性自动生成 getter 和 setter 方法。例如, 有这样一个类, 代码如下:

```
package com.fengmanfei.ch3;  
public class BeanDemo {  
    private String name;  
    private int id;  
}
```

这时, 选中类属性的两行, 然后右击, 在弹出的快捷菜单中选择“源代码” | “生成 getter 和 setter 方法”命令, 弹出如图 3.16 所示的对话框。

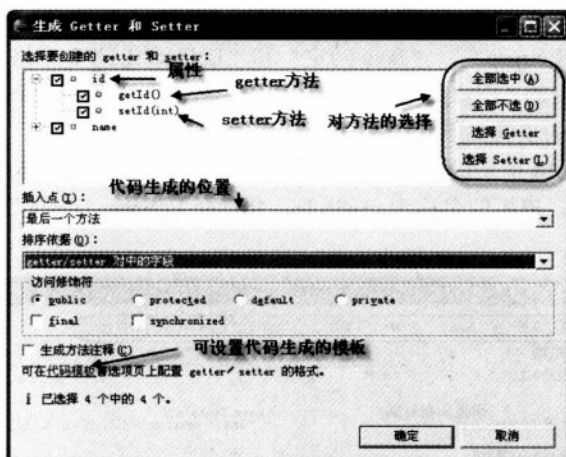


图 3.16 “生成 Getter 和 Setter”对话框

单击“确定”按钮后, 自动生成的代码如下:

```
package com.fengmanfei.ch3;  
public class BeanDemo {  
    private String name;  
    private int id;  
    public int getId() {  
        return id;  
    }  
    public void setId(int id) {  
        this.id = id;  
    }  
}
```

```
}  
public String getName() {  
    return name;  
}  
public void setName(String name) {  
    this.name = name;  
}  
}
```

3.3.4 代码的重构

重构对程序开发来说是个很重要的环节。尤其在后期维护时，对代码的修改更加重要。Eclipse 强大的重构功能，能使代码的修改变得简单许多。

使用重构功能时，只需右击所要修改的变量、方法或代码，在弹出的快捷菜单中选择“重构”命令，然后选择重构的种类即可。“重构”菜单如图 3.17 所示。



图 3.17 “重构”菜单

⚠注意：选择不同的代码区域，会出现不同的“重构”菜单。

Eclipse 的重构功能很有用处，希望读者能够好好掌握，这里就不再详细介绍了。

3.3.5 查看源代码

Eclipse 提供了快捷地查看源代码的方式。只需将鼠标移动到所要查看的类名、方法或变量上，同时按住 Ctrl 键，然后单击，便可直接查看源代码。如果此时要查看的类没有代码，可以关联源代码后再查看。以下以关联 JDK 的源代码为例进行讲解。

⚠注意：安装了中文语言包的 Eclipse 3.1 在查看源代码时会出现以下的错误，这可能是 Eclipse 语言包的一个 bug，如图 3.18 所示。

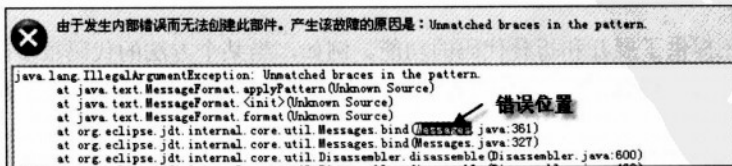


图 3.18 关联源文件时报错信息

其实, 仔细查看 Eclipse 的源代码可以发现 org.eclipse.jdt.internal.core.util. Messages 类中的 disassembler_opentypeddeclaration 字段, 问题出现在使用语言包时调用的 org.eclipse.jdt.core.nl1_3.1.1.jar 文件里面的 messages_zh.properties 文件, 在文件中找到这个字段, 显示为 disassembler_opentypeddeclaration=\ \u201C{\u201D, 而其他语言文件的内容是 disassembler_opentypeddeclaration=\ '{', 所以更改一下代码, 然后打包。修正好的语言包可以在本书附带的光盘中找到, 读者只需复制到对应的 plugins 文件夹下即可。

(1) 按照查看源代码的方法查看 String 这个类, 由于此时尚未关联任何代码, 所以会出现如图 3.19 所示的界面。

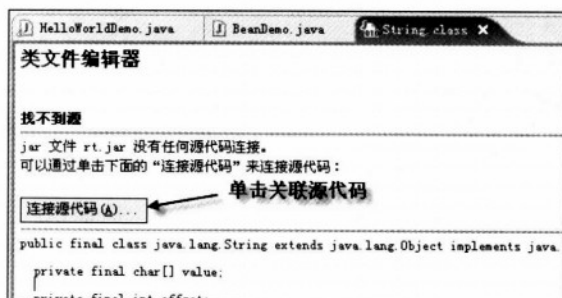


图 3.19 查看源代码

(2) 单击“连接源代码”按钮, 在弹出的对话框中作如图 3.20 所示的设置, 并单击“确定”按钮。

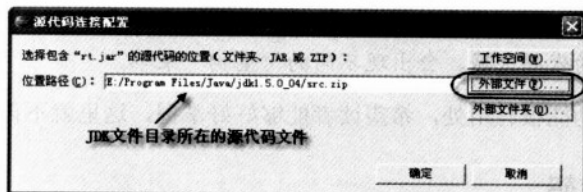


图 3.20 “源代码连接配置”对话框

注意: 若代码文件不是压缩文件而是文件夹, 也可以关联。这时需要单击“外部文件”按钮; 若源代码在项目工程内部的某个文件夹中, 这时需要单击“工作空间”按钮。

3.3.6 代码的展开和折叠

Eclipse 还提供了展开和折叠代码的功能。例如, 当某个方法的代码很多时, 可以折叠起来, 在查看时再将它打开, 这样可方便地查看代码部分。如图 3.21 所示为展开和折叠代码的示意图。

用户可以单击展开和折叠按钮来展开和折叠代码, 当代码处于折叠状态时, 只要将鼠标放到展开按钮上, 就会出现悬浮的方法体。

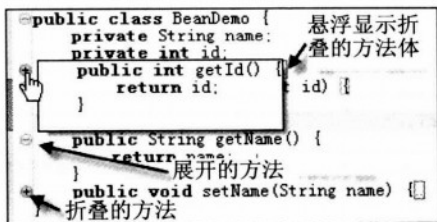


图 3.21 展开和折叠代码示意图

3.3.7 代码比较

对同一个文件，每次保存后，Eclipse 会自动保存一个文件备份，而且提供当前文件与历史文件对比的功能。这个功能对开发很有用，用户可以方便地找出修改了哪些地方。

(1) 在编辑区右击，在弹出的快捷菜单中选择“本地历史记录”|“比较对象”命令，弹出如图 3.22 所示的“将 Java 元素与本地历史记录进行比较”对话框。历史记录是按照时间来保存的，用户只需选择某个时间的历史记录即可。

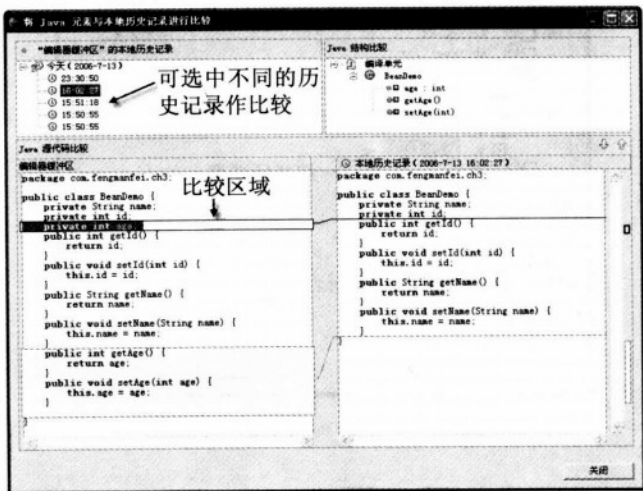


图 3.22 “将 Java 元素与本地历史记录进行比较”对话框

(2) 也可以将当前文件恢复到某一历史记录的文件，此时只需选择“本地历史记录”|“替换为”命令，在弹出的对话框中选择所要恢复的历史记录即可。

提示：对于存放在 CVS 服务器中的文件，也可以将本地的文件与服务器上的文件进行对比，这样方便了解小组中其他人对该类的改动情况。有关 CVS 的知识这里就不详细介绍了，有兴趣的读者可以参考相关资料。

(3) 要设置保存在本地历史记录的数量，可选择“窗口”|“首选项”|“常规”|“工作空间”|“本地历史记录”命令，弹出如图 3.23 所示的对话框。

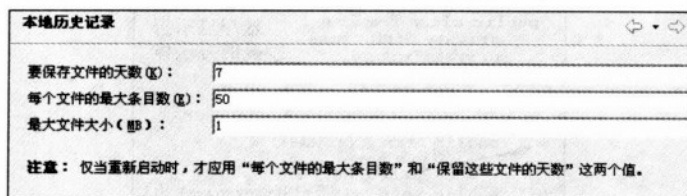


图 3.23 本地历史记录设置窗口

3.3.8 子类中覆盖父类的方法

子类覆盖父类的方法是面向对象设计很重要的方法。Eclipse 提供了方便覆盖父类方法的功能。在编辑区右击，在弹出的快捷菜单中选择“源代码”|“覆盖/实现方法”命令，弹出如图 3.24 所示的对话框。在这里选择所要覆盖的父类方法后，单击“确定”按钮，便可以自动生成该方法的代码，接下来的工作只需要添加方法体即可。

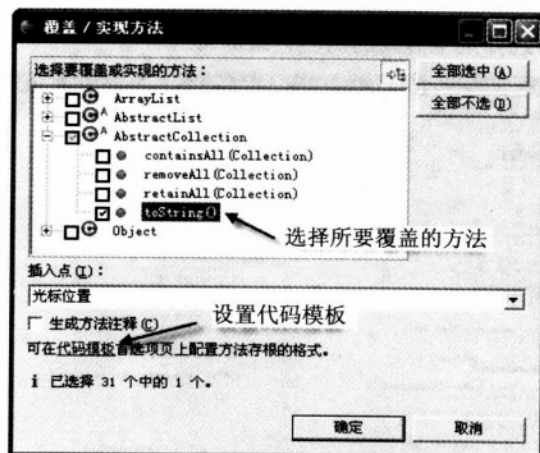


图 3.24 “覆盖/实现方法”对话框

3.4 代码错误提示

当编写的代码出现错误时，Eclipse 会及时反馈给用户，告诉用户哪里出了问题，是什么样的问题，如何修正等。通过使用这些功能，能快速地修复代码中的错误。

3.4.1 如何定位错误

编写程序时不可避免地要出现各种错误。Eclipse 的错误处理功能非常强大。如图 3.25 所示，当一个地方出现错误时，错误的提示会在界面上多个地方显示出来，这样就可以方

便地定位错误了。

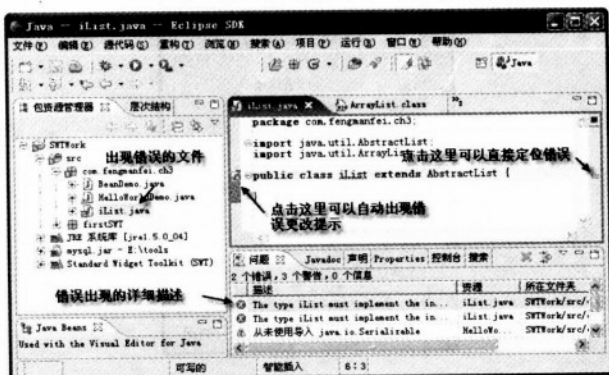


图 3.25 当有错误时的 Eclipse 界面

3.4.2 自动修正错误

为了显示可用于问题或警告的修正建议，编辑器的注释栏将显示“小灯泡”状态。单击出现错误的小灯泡，将会出现如图 3.26 所示的修正建议。

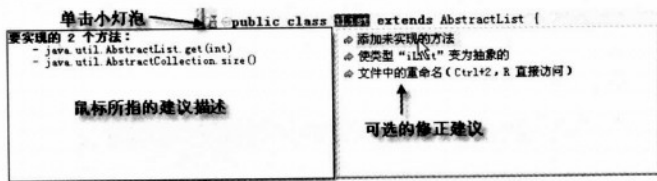


图 3.26 自动修正功能

不同的错误会出现不同的错误提示，用户可以自由进行选择。自动修正功能很好用，能大大加快更正错误的效率，希望读者在编写代码时能够深刻体会。

3.5 文件查找

查找也是开发中常用的功能之一。Eclipse 提供了多种查找的方式，包括文件内部的查找和项目内的查找等。

3.5.1 文件内部查找

查找功能是一项基本功能，Eclipse 提供了功能强大的查找功能。例如，在文件内部查找某个指定的字符串，此时，使用 Ctrl+F 快捷键调出如图 3.27 所示的“查找 / 替换”对话框即可。

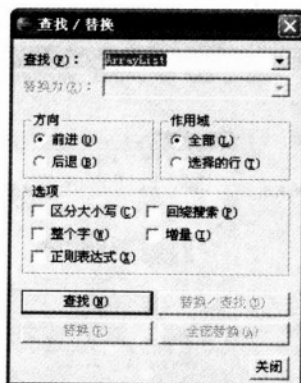


图 3.27 “查找/替换”对话框

技巧：一般来说，许多软件都支持 Ctrl+F 快捷键进行查找。例如，如果在网页中查找某个字符串，也可使用该快捷键调出查找的对话框。

3.5.2 项目内查找

另外，Eclipse 还可以设定查找目标进行搜索。选择“搜索”|“搜索”命令，打开“搜索”对话框，如图 3.28 所示。通过设置，就可以对不同类型的文件进行查找了，所以搜索功能也是 Eclipse 的亮点之一。

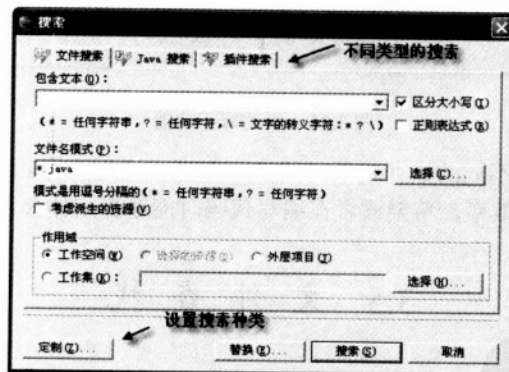


图 3.28 “搜索”对话框

3.6 使用快捷键

快捷键是程序开发者所常用的，记往常用的快捷键能大大提高开发效率。Eclipse 不仅可以使用内置的一些快捷键，也可以自定义快捷键。

3.6.1 显示快捷键说明

Eclipse 3.1 提供了快速显示快捷键的功能，在编辑区使用快捷键 Ctrl+Shift+L 即可在编辑区上显示所有的快捷键及其说明提示，如图 3.29 所示。

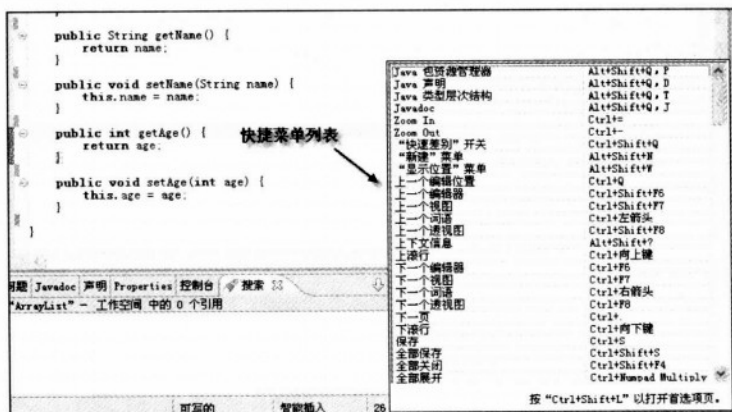


图 3.29 快捷键列表

3.6.2 自定义快捷键

使用系统内置的快捷键并不能满足用户的需求，用户有时还可按照自己的喜好来设置快捷键。Eclipse 也提供了自定义快捷键的功能。

选择“窗口”|“首选项”命令，在弹出的对话框中选择“常规”|“键”选项，此时，可以更改快捷键的设置，如图 3.30 所示。使用快捷键也是开发软件中提高效率的一种方式，读者应该记住一些常用功能的快捷键，例如格式化代码为 Ctrl+Shift+F、保存文件为 Ctrl+S 等。

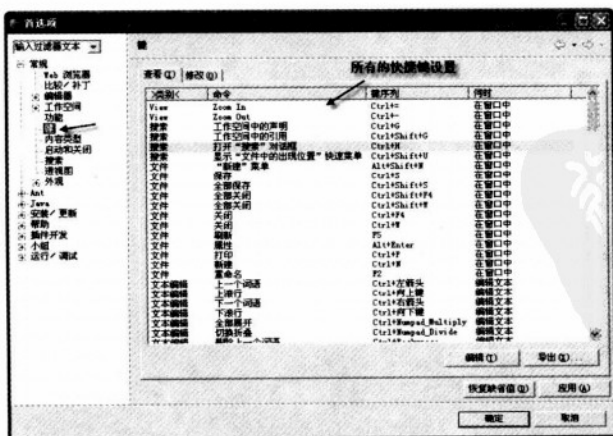


图 3.30 快捷键设置

3.7 本章小结

本章对 Eclipse 的开发环境进行了简单的介绍，所列举的一些功能都是开发中经常遇到的。如果读者想要深入了解 Eclipse 的开发环境，请参阅 Eclipse 的帮助文档，那里有详细的讲述。另外，对 Eclipse 的 CVS 小组开发功能、Debug 调试功能、Ant 构建和 JUnit 等本章并没有作详细的说明，这些都是开发中常用的，希望读者也能掌握。



第 2 篇



SWT 进阶篇

- 第 4 章 SWT 开发基础
- 第 5 章 SWT 基本组件
- 第 6 章 面板容器类
- 第 7 章 SWT 布局管理器
- 第 8 章 SWT 中的事件模型



第 4 章 SWT 开发基础

本章首先从一个典型的 SWT 程序入手,分析 SWT 的程序结构;然后深入讨论 SWT 程序的两个重要的对象——Display 对象和 Shell 对象;最后,对 SWT 程序所涉及的包作简要的介绍。

4.1 SWT 应用程序基本结构

首先通过一个示例窗口,从代码的角度看 SWT 应用程序的结构。以下代码就是一个典型 SWT 应用程序,该程序打开一个带有按钮的窗口,该程序运行后的效果如图 4.1 所示。



图 4.1 SWT 程序示例

创建一个 SWT 程序的步骤如下:

- (1) 在项目文件中的 src 文件上单击鼠标右键,在弹出的快捷菜单中选择“新建”|“类”命令,如图 4.2 所示。
- (2) 在弹出的“新建 Java 类”对话框中输入包名和类名,具体输入的信息如图 4.3 所示。然后单击“完成”按钮。



图 4.2 新建类

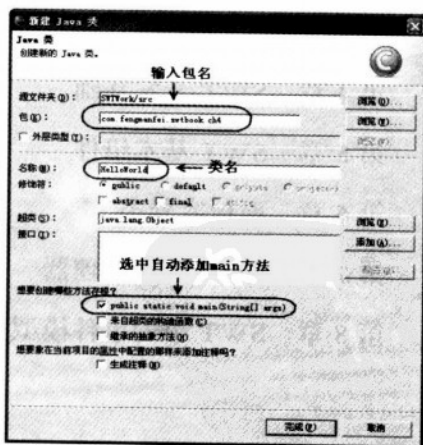


图 4.3 创建 HelloWorld 类

- (3) 在创建的类文件中输入以下代码:

HelloWorld.java

```
package com.fengmanfei.swtbook.ch4;

import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;

public class HelloWorld {
    public static void main(String[] args) {
        // 第一部分：初始化窗口
        Display display = new Display(); // 创建 Display 类的实例
        Shell shell = new Shell(display); // 创建该 Display 的 Shell 类实例
        shell.setText("SWT"); // 设置窗口标题
        // 第二部分：向 Shell 中添加 Widget 控件
        // 在这里填写在当前 Shell 中放置部件的代码
        Button button = new Button(shell, SWT.CENTER); // 创建一个按钮
        button.setText("欢迎进入到 SWT 的世界!"); // 设置按钮显示的文字
        button.pack();
        // 第三部分：GUI 调用
        shell.pack(); // 调整布局
        shell.open(); // 打开窗口
        while (!shell.isDisposed()) {
            if (!display.readAndDispatch())
                display.sleep();
        }
        // 销毁 Display 实例，释放创建 Display 时所获取的内存资源，断开与本地操作系统的连接
        display.dispose();
    }
}
```

一般来说，创建一个 SWT 程序有 3 个部分，如代码中所示。

(1) 初始化窗口：首先要创建 Display 对象和 Shell 对象，Display 对象封装了调用操作系统有关的方法。但若若要显示可视化的窗口，只有 Display 对象是不够的，还要创建 Shell 对象，该对象是窗口对象。Display 对象和 Shell 对象是学习 SWT 程序的两个非常重要的概念。以下的章节中会作详细的讲述。

(2) 添加控件：在 Shell 窗口对象中添加各种控件（Widget）。所谓控件就是所说的按钮、文本框、选项卡等，这些控件对象都在 org.eclipse.swt.widgets 和 org.eclipse.swt.custom 包下。

(3) GUI 调用：首先调用 shell.open() 方法打开窗口，并同时显示出该窗口中的所有控件。最后的循环表示只要 Shell 窗口还未释放，Display 对象就会调用 readAndDispatch() 方法来跟踪事件队列中用户所注册的事件。一旦用户关闭了窗口，则调用 Display 对象的 dispose() 方法释放 Display 对象。

综上所述，创建一个 SWT 程序，都是以这种结构形式使用的，其中第一部分和第三部分类似，只是在第二部分添加控件的代码有所不同。

4.2 Display 类

4.2.1 Display 类概述

Display 类是 SWT 应用程序中的基础类，它负责在应用程序和本地操作系统之间建立交互。Display 类是从 Device（设备）继承而来，同时继承的还有 Printer（打印机）类。所以要进行打印操作就要使用 Printer 类，Printer 类会在后面的章节中讲述。Display 类的类继承关系如图 4.4 所示。

一般来说，一个应用系统只要创建一个 Display 的实例即可。创建 Display 实例的同时也就启动了一个专有的线程，该线程将执行事件循环，并且保持 UI 线程。SWT 应用中所有的本地部件界面调用都是在这个线程中完成的，如果同时有其他 UI 线程试图访问这些本地部件，都会抛出 SWT.ERROR_THREAD_INVALID_ACCESS 异常。

在 Display 的代码中会发现使用最多的是 OS 类，该类位于 org.eclipse.swt.internal.win32 包下，OS 类中使用许多 Native Method（本地方法），这里涉及了 JNI（Java™ Native Interface）的有关知识，读者有兴趣可以参考一下这方面的资料。总之，OS 类封装了大部分与本地操作系统的调用，是最底层的封装。如图 4.5 所示为 Display 在 SWT 应用程序中的结构。

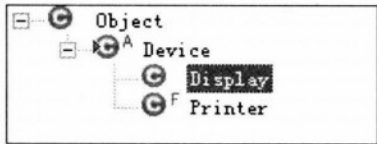


图 4.4 Display 类继承关系图

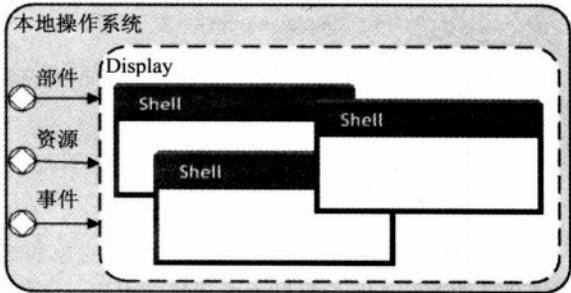


图 4.5 Display 对象在 SWT 程序中的结构

如图 4.5 所示，Display 对象封装了对本地操作系统资源、事件和各种控件的管理，是开发 SWT 应用程序的基础，所以有必要对 Display 类作详细介绍。

4.2.2 Display 类常用方法

Display 类的构造方法，如表 4.1 所示，该类有两个构造方法。

表 4.1 Display 类的构造方法

构造方法	说明
Display()	创建一个 Display 实例，并且创建一个 UI 线程。也可以通过 Display 类的静态方法 Display.getDefault()来创建一个 Display 实例
Display(DeviceData data)	使用 DeviceData 对象来创建 Display 实例，来调试错误信息

Display 类中常见的方法如表 4.2 所示。

表 4.2 Display 类的常用方法

常用方法	说 明
<code>void addFilter(int eventType, Listener listener)</code>	注册事件。当指定 <code>eventType</code> 类型的事件发生时，调用指定的事件处理（ <code>listener</code> ）。该方法所添加的事件类型和事件处理方法在其他事件触发之前调用，所以有可能影响到其他事件调用。该方法适合作调试时使用
<code>void addListener(int eventType, Listener listener)</code>	注册事件。当指定 <code>eventType</code> 类型的事件发生时，调用指定的事件处理（ <code>listener</code> ）。当事件触发时调用 <code>listener</code> 对象的 <code>handleEvent()</code> 方法来处理
<code>public void asyncExec(Runnable runnable)</code>	提供了在 SWT 类中创建线程的方法，UI 线程会调用 <code>runnable</code> 对象的 <code>run()</code> 方法。用此方法建立的线程与 UI 线程异步。注意区别 <code>syncExec(Runnable runnable)</code> 方法
<code>public void beep ()</code>	发出嘟嘟声（需硬件支持）
<code>public void close ()</code>	关闭 SWT 与底层操作系统的连接
<code>public void disposeExec (Runnable runnable)</code>	当销毁 Display 对象前执行 <code>runnable</code> 对象中的 <code>run()</code> 方法
<code>static Display findDisplay (Thread thread)</code>	返回指定 UI 线程所在 Display 对象，如果该线程不是 UI 线程，将返回 <code>null</code>
<code>Widget findWidget(int handle)</code>	返回指定句柄（ <code>handle</code> ）的 Widget 控件对象，如果没有找到则返回 <code>null</code>
<code>public Widget findWidget (int handle, int id)</code>	返回指定句柄（ <code>handle</code> ）和 <code>id</code> 的 Widget 控件对象，如果没有找到则返回 <code>null</code>
<code>public Shell getActiveShell ()</code>	返回当前激活状态的面板 Shell 对象，如果当前没有激活状态的 Shell 对象，将返回 <code>null</code>
<code>public Rectangle getBounds ()</code>	返回当前 Display 对象的大小和位置
<code>public Rectangle getClientArea ()</code>	返回显示数据的大小和位置
<code>static Display getCurrent()</code>	返回当前 UI 线程的 Display 对象，如果当前运行的线程不是 UI 线程，则返回 <code>null</code>
<code>public Point getCursorLocation ()</code>	返回当前鼠标的屏幕位置，以左上方为原点
<code>public Point [] getCursorSizes ()</code>	返回鼠标大小的数组
<code>Object getData(String key)</code>	返回指定 <code>key</code> 的对象，如果该对象不存在，则返回 <code>null</code>
<code>static Display getDefault()</code>	返回默认的 Display 对象。如果还未创建 Display 对象，则创建 Display 对象，并标记该线程为 UI 线程
<code>public int getDismissalAlignment ()</code>	返回对话框中按钮的对齐方式。值可为 <code>SWT.LEFT</code> 或 <code>SWT.RIGHT</code>
<code>public int getDoubleClickTime ()</code>	返回双击鼠标的的时间间隔。以毫秒为单位
<code>public Control getFocusControl ()</code>	返回获得焦点的控件，如果没有获得焦点的控件，则返回 <code>null</code>
<code>public boolean getHighContrast ()</code>	如果使用了高亮显示方式，则返回 <code>true</code> ，否则返回 <code>false</code>
<code>public int getIconDepth ()</code>	返回图标色彩深度的最大值
<code>public Point [] getIconSizes ()</code>	返回图标大小的数组

续表

常用方法	说 明
<code>public Monitor [] getMonitors ()</code>	返回显示器数组, Eclipse 3.0 开始支持带有多显示器的设备, 有些笔记本电脑能够同时将桌面在内屏和外屏上同时显示
<code>public Monitor getPrimaryMonitor()</code>	返回主显示器
<code>public Shell[] getShells()</code>	返回该 Display 对象所有的未销毁 Shell 对象
<code>public Thread getSyncThread()</code>	返回 <code>syncExec()</code> 方法设置的线程对象, 如果没有运行该线程, 则返回 <code>null</code>
<code>public Color getSystemColor(int id)</code>	返回指定颜色常量 (SWT 类中) 所对应的颜色 Color 对象, 如果没有找到, 则返回黑色。这种方式获得的颜色对象不需要被释放, 因为它是系统提供的
<code>public Cursor getSystemCursor(int id)</code>	返回指定光标常量 (SWT 类中) 所对应的光标 Cursor 对象, 如果没有找到常量所对应的光标对象, 则返回 <code>null</code> 。同样, 该种方式获得 Cursor 对象也不需要释放 SWT 类中的光标常量有: SWT.CURSOR_ARROW, SWT.CURSOR_WAIT, SWT.CURSOR_CROSS, SWT.CURSOR_APPSTARTING, SWT.CURSOR_HELP, SWT.CURSOR_SIZEALL, SWT.CURSOR_SIZENESW, SWT.CURSOR_SIZENS, SWT.CURSOR_SIZENWSE, SWT.CURSOR_SIZEWE, SWT.CURSOR_SIZE, SWT.CURSOR_SIZES, SWT.CURSOR_SIZEE, SWT.CURSOR_SIZEW, SWT.CURSOR_SIZENE, SWT.CURSOR_SIZESE, SWT.CURSOR_SIZESW, SWT.CURSOR_SIZENW, SWT.CURSOR_UPARROW, SWT.CURSOR_IBEAM, SWT.CURSOR_NO, SWT.CURSOR_HAND
<code>public Font getSystemFont()</code>	返回系统所有的字体 (Font) 对象数组。同样, 该种方式获得 Font 对象也不需要释放。通常创建控件时不需要指定字体, 将会按默认的字体系创建控件
<code>public Image getSystemImage(int id)</code>	返回指定图标常量 (SWT 类中) 所对应的图标 Image 对象, 如果没有找到, 则返回 <code>null</code> 。这种方式获得的 Image 对象也不需要被释放 SWT 类中的图标常量有: SWT.ICON_ERROR, SWT.ICON_INFORMATION, SWT.ICON_QUESTION, SWT.ICON_WARNING, SWT.ICON_WORKING
<code>public Tray getSystemTray()</code>	返回系统托盘
<code>public Thread getThread()</code>	返回当前的 UI 线程
<code>public Point map(Control from,Control to,Point point)</code>	将点 (Point 类型) 的坐标从一个控件映射到另一个控件
<code>public Point map(Control from,Control to,int x,int y)</code>	将指定点的坐标从一个控件映射到另一个控件

续表

常用方法	说 明
<code>public Rectangle map(Control from,Control to,Rectangle rectangle)</code>	将区域 (Rectangle 类型) 从一个控件映射到另一个控件
<code>public Rectangle map(Control from,Control to,int x,int y,int width,int height)</code>	将区域从一个控件映射到另一个控件
<code>public boolean post(Event event)</code>	触发底层事件, 比如鼠标或键盘事件。在测试程序时可以模仿用户操作的事件, 一般 SWT 程序不会用到该方法
<code>public boolean readAndDispatch()</code>	读取操作系统事件队列中的事件, 如果队列事件中还有事件未处理, 则返回 <code>true</code> ; 如果该 UI 线程处于等待状态 (Sleep), 则返回 <code>false</code> 。该方法还负责检查由 <code>syncExec()</code> 或 <code>asyncExec()</code> 创建的线程
<code>protected void release()</code>	释放该 Display 对象的系统资源, 调用该方法 <code>isDisposed()</code> 返回 <code>true</code>
<code>public void removeFilter(int eventType, Listener listener)</code>	移除由 <code>addFilter(int eventType, Listener listener)</code> 方法注册事件
<code>public void removeListener(int event Type, Listener listener)</code>	移除由 <code>addListener (int eventType, Listener listener)</code> 方法注册事件
<code>public void setCursorLocation(int x,int y)</code>	设置鼠标的坐标, 以横坐标 (x) 和纵坐标 (y) 为参数
<code>public void setCursorLocation(Point point)</code>	设置鼠标的坐标, 以 Point 对象为参数
<code>public void setData(String key,Object value)</code>	设置指定键(key)和值(value)。设置的值可通过 <code>getData(String key)</code> 来获得
<code>public void setData(Object data)</code>	设置数据
<code>public static void setAppName(String name)</code>	设置 SWT 应用系统的名称
<code>public void setSynchronizer(Synchronizer synchronizer)</code>	设置与 Display 对象同步的对象
<code>public boolean sleep()</code>	UI 线程是否处于等待状态
<code>public void syncExec(Runnable runnable)</code>	提供了在 SWT 类中创建线程的方法, 用此方法建立的线程与 UI 线程同步
<code>public void timerExec(int milliseconds, Runnable runnable)</code>	在指定时间 (以毫秒为单位) 后, 创建线程。如果事件指定为负数, 则线程将不会启动
<code>public void update()</code>	重画 Display 对象中的控件
<code>public void wake()</code>	唤醒 UI 线程

⚠注意: Display 中涉及有关线程的问题将会在下文中作详细的介绍。

4.3 Shell 类

4.3.1 Shell 类概述

Shell 类是显示在桌面上的窗口, 可以是顶级窗口 (Top Level Shells) 或者是对话框

(Secondary Or Dialog Shells) 窗口，如图 4.6 所示为创建的窗口示例。



图 4.6 窗口示例

窗口有最大化、最小化和还原 3 种状态。当窗口最大化时，窗口会充满整个屏幕；当窗口最小化时，窗口会隐藏到任务栏中，以图标的方式显示。无论是最大化还是最小化，此时都可以选择还原到初始状态。

- 顶级窗口：创建 Shell 对象时传入的是 Display 对象，则为顶级窗口。代码如下：

ShellDemo.java

```
//传入的是 Display 对象，为顶级窗口
//设置窗口的样式为 SWT.SHELL_TRIM
Shell topShell = new Shell(display, SWT.SHELL_TRIM);
topShell.setText("顶级窗口");//设置标题
topShell.setSize(300,200);//设置窗口大小
topShell.open();
```

- 对话框窗口：创建 Shell 对象时传入的是 Shell 对象，则为对话框窗口。代码如下：

ShellDemo.java

```
//传入的是 Shell 对象，为对话框窗口
//设置窗口的样式为 SWT.DIALOG_TRIM
Shell dialogShell = new Shell(topShell, SWT.DIALOG_TRIM );
dialogShell.setText("对话框窗口");//设置标题
dialogShell.setSize(200,150);//设置窗口大小
dialogShell.open();
```

4.3.2 不同窗口的样式

通过使用 org.eclipse.swt.SWT 类中的常量可以构造不同的窗口样式，如可以禁用最大化按钮或者只显示标题等。例如，创建一个只显示标题栏的窗口所编写的代码如下：

```
Shell shell = new Shell(display, SWT.TITLE);
```

编写一个只能够最大化窗口的代码如下：

```
Shell shell = new Shell(display, SWT.MAX);
```

以此类推，创建不同样式的窗口只需改变 SWT 中的常量即可。表 4.3 显示的为不同样

式窗口的示例。

表4.3 Shell类的不同样式

样 式 常 量	说 明	图 例
SWT.TITLE	只显示标题栏，去掉了“关闭”、“最大化”和“最小化”按钮	
SWT.CLOSE	只显示“关闭”按钮	
SWT.MIN	只显示“最小化”按钮和“关闭”按钮	
SWT.MAX	只显示“最大化”按钮和“关闭”按钮	
SWT.BORDER	显示边框	
SWT.RESIZE	窗口的大小可以通过鼠标拖动来设定	
SWT.NO_TRIM	既无边框也无标题	
SWT.SHELL_TRIM	相当于 CLOSE TITLE MIN MAX RESIZE 同时显示	
SWT.DIALOG_TRIM	相当于 TITLE CLOSE BORDER 同时显示	

除了表中提供的这些样式外，还可以选择使用 SWT.APPLICATION_MODAL、SWT.MODELESS、SWT.PRIMARY_MODAL 和 SWT.SYSTEM_MODAL。这4个样式是设定窗口的行为模式，因不同的操作系统而异。这4个样式一般都与其他样式连用，例如：

```
Shell shell = new Shell(display, SWT.SHELL_TRIM|SWT.SYSTEM_MODAL);
```

4.3.3 应用多个样式

如果想使一个窗口具备多个样式，在创建 Shell 对象时可以在多个样式中使用“|”运算符。例如，创建一个只显示“关闭”按钮的窗口，并且该窗口能够改变大小，编写的代码如下：

```
Shell shell = new Shell(display, SWT.CLOSE|SWT.RESIZE);
```

⚠注意：SWT 中各种控件选择多个样式时都是以这种方式编写的，以“|”为分割符。

4.3.4 Shell 类的主要方法

理解了 Shell 的意义，下面来看一下 Shell 类的具体的方法。图 4.7 所示为 Shell 类的继承结构图。从图中可以看到 Shell 类继承自几个重要的类——Control 类、Composite 类和

Canvas 类。这几个类将会详细地在下文中讲述。

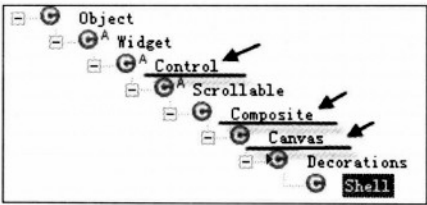


图 4.7 Shell 类的继承结构图

首先看一下构造方法，表 4.4 所示为 Shell 类的构造方法。

表4.4 Shell类的构造方法

构造方法	说 明
Shell()	无参构造方法，默认 Display 对象为 null
Shell(Display display)	传入 Display 对象
Shell(Display display, int style)	传入 Display 对象并指定样式
Shell(int style)	传入样式
Shell(Shell parent)	传入父窗口 Shell 对象
Shell(Shell parent, int style)	传入父窗口 Shell 对象并且指定样式

表 4.5 所示为 Shell 类的常用方法。

表4.5 Shell类的常用方法

构造方法	说 明
void addShellListener(ShellListener listener)	为窗口注册 ShellListener 事件
void forceActive ()	使该窗口为激活状态
Rectangle getBounds ()	返回该窗口的边界值类型为 Rectangle
boolean getEnabled ()	是否已启用
int getImeInputMode ()	返回窗口的输入编辑模式，返回值有 SWT.NONE、SWT.ROMAN、SWT.DBCS、SWT.PHONETIC、SWT.NATIVE 或 SWT.ALPHA
Point getLocation ()	返回窗口相对于父窗口所在位置的点，如果没有父窗口则返回相对于 Display 对象的点
Point getMinimumSize ()	返回窗口的最小值，x 表示窗口的最小宽度，y 表示窗口的最小高度
Region getRegion ()	返回窗口所定义的形状，如果没有设定形状，则返回 null
Shell getShell ()	返回窗口对象
Shell [] getShells ()	返回该窗口下所有的子窗口
Point getSize ()	返回窗口的宽度和高度， x 表示宽度， y 表示高度
boolean isEnabled ()	是否可用，如不可用则灰色显示
boolean isVisible ()	是否可见
void open ()	打开口，并且设置该窗口为激活状态

续表

构造方法	说 明
<code>removeShellListener (ShellListener listener)</code>	取消 <code>ShellListener</code> 事件
<code>void setActive ()</code>	设置该窗口为激活状态
<code>void setActiveControl (Control control)</code>	设置当前活动的控件
<code>void setEnabled (boolean enabled)</code>	设置可用
<code>void setImeInputMode (int mode)</code>	设置输入编辑模式, 参考 <code>getImeInputMode ()</code> 方法
<code>void setMinimumSize (int width, int height)</code>	设置最小的窗口宽度和高度
<code>void setMinimumSize (Point size)</code>	设置最小的窗口宽度和高度
<code>void setRegion (Region region)</code>	设置窗口的 <code>Region</code> 对象, 当需要自定义窗口形状时使用
<code>void setVisible (boolean visible)</code>	设置窗口可见状态

4.3.5 创建多个窗口

在程序设计的过程中, 经常会遇到在一个父窗口中同时打开多个窗口的情况。如图 4.8 所示为创建多个窗口的程序运行后的预览图。当单击“创建子窗口”按钮时, 将创建一个新窗口。

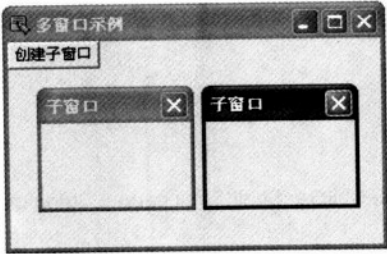


图 4.8 创建多个窗口

该程序的代码如下:

MultiShell.java

```
package com.fengmanfei.swtbook.ch4;

import org.eclipse.swt.SWT;
import org.eclipse.swt.events.SelectionAdapter;
import org.eclipse.swt.events.SelectionEvent;
import org.eclipse.swt.graphics.Image;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;

public class MultiShell {

    public static void main(String[] args) {
```

```

Display display = new Display();

final Shell parent = new Shell(display, SWT.SHELL_TRIM);
parent.setText("多窗口示例 ");
parent.setSize(300, 200);
//设置父窗口图标
parent.setImage( new Image( display , "F:\\icon\\edit.gif"));

Button addShell = new Button(parent, SWT.CENTER);
addShell.setText("创建子窗口");
//注册按钮单击事件
addShell.addListener(new SelectionAdapter() {
    public void widgetSelected(SelectionEvent event) {
        //当单击时创建子窗口
        createChildrenShell(parent, "子窗口");
    }
});
addShell.pack();

parent.open();
while (!parent.isDisposed()) {
    if (!display.readAndDispatch())
        display.sleep();
}
display.dispose();
}


public static Shell createChildrenShell(Shell parent, String childrenName) {
    //创建子窗口
    Shell shell = new Shell(parent, SWT.DIALOG_TRIM);
    shell.setText(childrenName);
    shell.setSize(100, 100);
    shell.open();
    return shell;
}
}

```

该程序通过 createChildrenShell()方法创建子窗口，这样的好处是减少了重复的代码。另外程序中还设置了窗口的图标，是通过以下语句来实现的：

```
parent.setImage( new Image( display , "F:\\icon\\edit.gif"));
```

其中，"F:\\icon\\edit.gif"是图标所在的绝对路径。

 **注意：**该程序的代码中涉及到了按钮单击事件，有关控件的事件处理将在以后的章节讲解。

4.4 SWT 包类结构

通过 Eclipse 自带的帮助系统，可以查阅相关的 SWT API 参考。在 Eclipse 工作平台上选择“帮助”|“帮助内容”命令，弹出的窗口即为 Eclipse 的帮助窗口。如图 4.9 所示为 SWT API 参考帮助窗口。

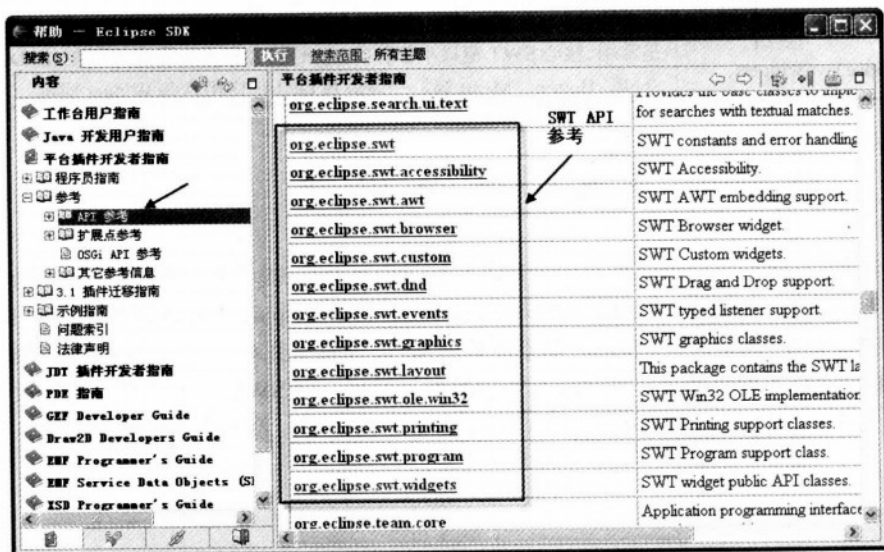


图 4.9 SWT API 参考帮助窗口

表 4.6 所示为每个 SWT 包的具体介绍。

表 4.6 SWT 包的具体介绍

包 名	功 能 描 述
org.eclipse.swt	SWT、SWTException 和 SWTError 类，提供了 SWT 常量与对异常处理的支持，SWT 在构造窗口组件和事件处理中有很大的用途
org.eclipse.swt.accessibility	Eclipse 易访问性包，专门为残疾人设计
org.eclipse.swt.awt	提供了在 SWT 程序中嵌入 AWT 程序的接口
org.eclipse.swt.browser	提供了实现浏览器功能的控件
org.eclipse.swt.custom	除了对应本地的控件外，该包提供了一些自定义的控件
org.eclipse.swt.dnd	提供了对拖放（Drag and Drop）的支持
org.eclipse.swt.events	提供了对 SWT 事件监视器（Event Listener）的支持，如 Button 的 Selection Listener、Mouse 的 MouseListener、MouseMoveListener 和 MouseTrackListener 等，还有与这些对应 Listener 的默认 Adapter 实现类和 Event 类
org.eclipse.swt.graphics	包含了 SWT 中 graphic 类，如 Color、Font 和 Image 等

续表

包 名	功 能 描 述
org.eclipse.swt.layout	控制 GUI 程序 Layout 的类所在（当然包括了相关的结构数据类），其中有 FillLayout、GridLayout 和 RowLayout 以及 FormLayout 等
org.eclipse.swt.ole.win32	提供了 SWT 中 Win32 OLE 实现的一些类
org.eclipse.swt.printing	支持打印机的类
org.eclipse.swt.program	支持在 SWT 程序中使用其他应用程序打开文件
org.eclipse.swt.widgets	是常用、核心 SWT 窗口小部件(widget)的公有 API 类定义所在。如 Display、Shell、Button、Menu 等。一般编写 GUI 程序用这些 Widget 即可

4.5 本章小结

本章主要介绍了一个 SWT 程序的基本结构，也是开发 SWT 程序的基础。然后对 SWT 两个重要的类——Display 和 Shell 类作了详细介绍，其中 Display 类是管理本地资源的桥梁，理解该类是开发 SWT 程序的基础。随着不断的深入，读者将会对该类的使用更加清晰。Shell 是 SWT 最底层的窗口实现，也是最基本的控件。最后，对 SWT 程序所涉及的包做了一下了解，尤其是 Eclipse 自带的帮助系统，是很好的学习资料，希望读者能充分利用。



第 5 章 SWT 基本组件

本章将首先从总体上来介绍各种组件，以及组件之间的继承关系。然后将对几个最常用的组件进行详细介绍，包括常见的样式和方法等。在讲述组件的同时也逐步渗透一些布局 and 事件处理的方法，为以后的学习打好基础。

5.1 SWT 控件类概述

SWT 作为一个 GUI 类库，其包含的类非常多。SWT 有一套非常优秀的体系框架，所以在了解这个框架之前，首先了解几个概念。

5.1.1 窗口小部件：Widget

部件是一个图形用户接口的元素，就是 GUI 应用中的按钮、文本框、选择框等经常使用的界面元素，这些部件可以响应事件与用户交互。

部件的外观是通过绘制显示出来的，通过绘制操作来维持部件的状态。用户用鼠标和键盘可以改变部件的状态，程序中的代码也可以改变部件的状态。当部件状态改变时，不管这种改变是由用户还是程序代码触发的，部件都会自动重绘以显示新状态，这是所有部件都有的重要特性。

实际上这相当于设置了部件一个通用的属性，并且不需要告诉部件该属性已经发生变化而需重绘以反映发生的变化，它会自动完成。简单地说，每个部件所在屏幕中呈现出来的效果总是不断地绘制出来的。

5.1.2 Widget 的继承关系

Widget 类是各种用户界面元素如按钮、列表、树和菜单整个继承体系的父类。不论是在 Widget 包里面还是以外的部件子类，都是通过实现事件机制以及增加特定的 API 来扩展 Widget 的基本行为。Widget 之间的容器继承关系是用父子关系来定义的。一个 Widget 的容器继承关系在它创建时就已确立，即构造器的 parent 参数，这种关系的建立是在运行时动态建立的。

在 SWT 中创建部件的方式与用构造方法创建一个普通的 Java 类不同。Widget 是抽象类，所以不能通过 new 关键词来创建一个 Widget 实例。在对类 Widget 的讨论中，实际指的是 Widget 类的子类，这是因为 Widget 类的子类共享相同的构造方法，这使得创建 Widget

保持高度一致性（实际上这是 Compound 复合设计模式的应用）。Widget 不能没有一个父类部件而存在，并且 Widget 一旦创建后，父类部件就不能改变了。这样，当一个部件被实例化时，操作系统的资源被 Widget 统一获得，这简化了 SWT 的实现，允许大部分部件驻留在操作系统，这样可以提高性能并减少内存占用。

如图 5.1 所示为几个父类部件类的继承关系图。

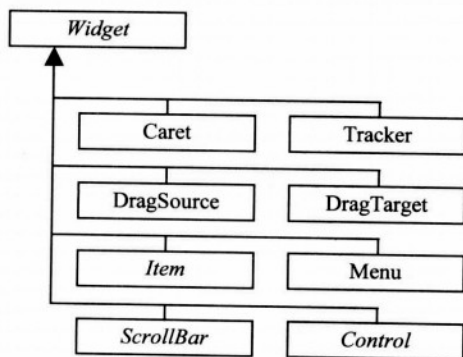


图 5.1 几个父类部件类的结构图

5.1.3 SWT 中的子类

Java 语言中可以使用修饰符 `final` 来标记一个类为不能被继承的类，在早期版本的 SWT 中，对于不能被继承的部件就是这样做的。然而对于一个不断开发当中的开源项目来说，这样会导致一些冲突，例如，如果在 SWT 发现一个部件类有问题，那么只有 SWT 开发团队可以修复它，没有其他方法可以通过扩展那个有问题的类，而覆盖有问题的方法来暂时修复。于是在现在的 SWT 版本中，部件会包含一个 `checkSubclass()` 方法，既可以允许修复错误，又不需要移除子类里面所有的限制。

现在如果需要定义一个部件不能被继承，可以通过覆盖 `checkSubclass()` 方法来达到目的。当一个类不允许子类化时，`checkSubclass()` 就会抛出一个 `SWTException("Subclassing not allowed")` 异常。在继承允许被继承的部件时，也仅限于可控的几点，特别是用来实现自定义 Widget 组件（Composite）和它的子类画布（Canvas）。为了指出这一点，Composite 里面的 `checkSubclass()` 方法没有强迫允许扩展。如果需要在 SWT 中生成一个新的 Widget，典型地需要扩展画布（Canvas）类，接下来用事件监听器来给它特定的外观和行为。在 SWT 中，用户接口是由 Widget 的实例组合而成，事件监听器被加到 Widget 上以运行特定的事件代码，而不是覆盖一个方法，程序员用监听器而不是子类化来反映用户接口的变化。在扩展可以被扩展的部件时，仍然不能参考或索引超类的内部细节，因为它们在不同平台和不同版本的 SWT 中还是有很明显的不尽相同。

所以一般在使用 SWT 中的各种部件时，尽量不要采用继承的方式创建自定义的部件。但在满足不了需要而必须创建自定义时，要倍加小心。

5.1.4 控件 (Controls) 与面板 (Composites)

一部分用户界面元素可以用控件来统一表示，它包含在之前介绍过的 Shell 窗口中。控件在用户接口中很普遍，按钮、进度条和表格都是控件。用户对操作系统的标准控件集很熟悉，在 SWT 中，操作系统控件被定义为抽象类 Control 的子类。例如 Button 类、Text 类，它们都是 Control 的子类，也都是操作系统控件。

前面说过，每个控件都有一个父部件，而这个父部件可以是 Composite 类或它的子类，Shell 类代表应用程序的顶层窗口，它也是 Composite 类的子类。Shell 在 Display 上被创建，Display 则代表显示屏幕。一个 Display 包含许多顶层窗口 Shell，每一个 Shell 都是 Composites 和 Controls 的组合树的根类，Composites 可以包含其他 Composites，允许这个树有任意深度。其体系结构如图 5.2 所示。

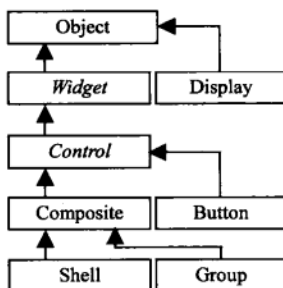


图 5.2 控件体系结构图

5.1.5 Widgets 不是 Controls

上文提到“一部分用户界面元素可以用控件来统一表示”。这说明组成 SWT 程序的一些用户接口元素并不能用 Controls 来表示，它们是一些对于控件完整性以及用户的需求来说非常必要的对象，但是操作系统却没有相应的类库支持。如树的条目 (TreeItem)，操作系统无法像其他控件一样描述一个 TreeItem，因为这些子项目在行为上不像控件，无法像其他控件一样在 SWT 的类库里面为其找到相应的 UI 模型。不是控件的 Widgets 集合是那些在所有操作系统都不能模型化为控件集合，这些控件就需要 SWT 依据不同的操作系统，自己绘制出来。图 5.3 是比较完整的部件体系结构图。

在清楚了 SWT 类库中的体系结构之后，结合 SWT 类的继承关系以及 SWT 的 GUI 体系框架，从另外一个更全面的角度来重新分析一下 SWT 应用程序。在此之前，需要创建一个稍微复杂一些的应用程序窗口，该窗口内包含分组 (Group)、按钮 (Button)、单选钮，如图 5.4 所示。

下面是该窗口的实现代码。其中关于部件和布局的内容会在以后的章节里详细介绍。

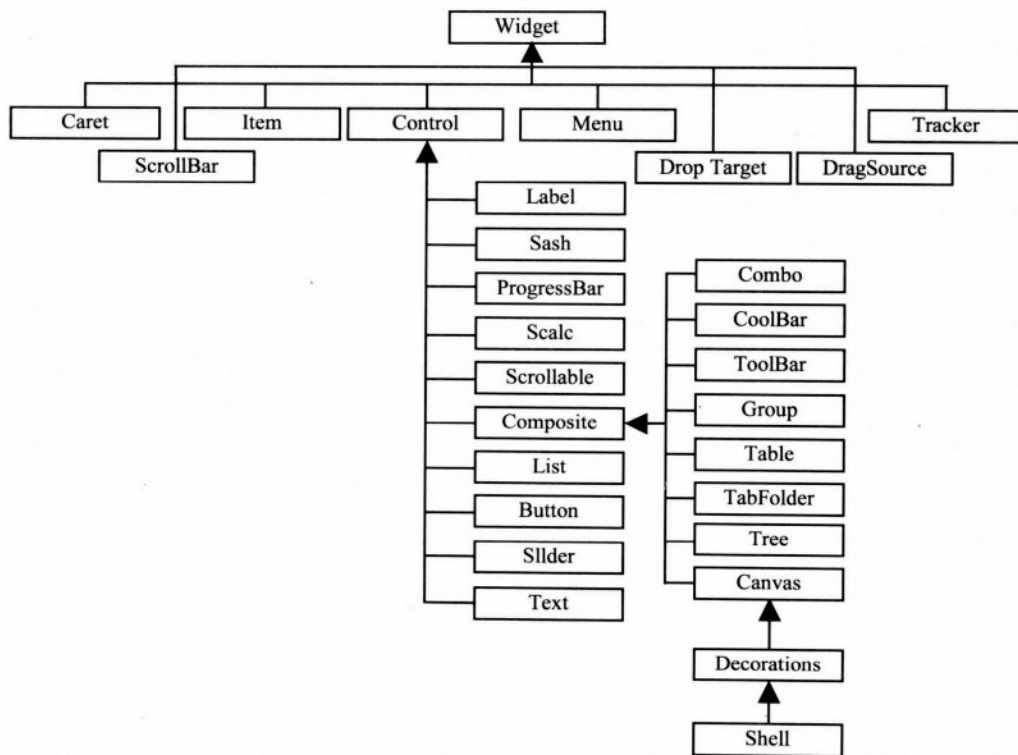


图 5.3 完整的部件体系结构图

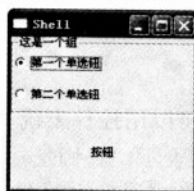


图 5.4 稍复杂一些的 SWT 应用程序窗口

ComplexSWT.java

```

package com.fengmanfei.ch5;

import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.FillLayout;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Group;
import org.eclipse.swt.widgets.Shell;

public class ComplexSWT {
    public static void main(String[] args) {

```

```
Display display = new Display();
// 获得该 Display 的一个 Shell 类的实例
Shell shell = new Shell(display);
// 为窗口设置大小
shell.setSize(200, 200);
// 为窗口设置标题栏文字
shell.setText("Shell");
// 为窗口设置布局类型
shell.setLayout(new FillLayout(SWT.VERTICAL));
// 在当前窗口中创建分组
Group group = new Group(shell, SWT.SHADOW_ETCHED_OUT);
// 为分组设置标题栏内容
group.setText("这是一个组");
// 为分组设置布局类型
group.setLayout(new FillLayout(SWT.VERTICAL));
// 在当前分组中创建单选钮 1
Button radio1 = new Button(group, SWT.RADIO);
// 为单选钮 1 添加说明文字
radio1.setText("第一个单选钮");
// 在当前分组中创建单选钮 2
Button radio2 = new Button(group, SWT.RADIO);
// 为单选钮 2 添加说明文字
radio2.setText("第二个单选钮");
// 在当前窗口中创建普通按钮
Button button = new Button(shell, SWT.PUSH);
// 为普通按钮添加说明文字
button.setText("按钮");
// 使分组的布局生效
group.layout();
// 使窗口的布局生效
shell.layout();
// 打开窗口，并且将窗口显示在屏幕上
shell.open();
while (!shell.isDisposed()) {
    if (!display.readAndDispatch())
        display.sleep();
}
// 销毁 Display 实例，释放创建 Display 时所获取的内存资源，断开与本地操作系统的连接
display.dispose();
}
```

现在通过图 5.5 来了解 SWT 应用程序窗口中的所有部件与类继承体系框架之间的对照关系。

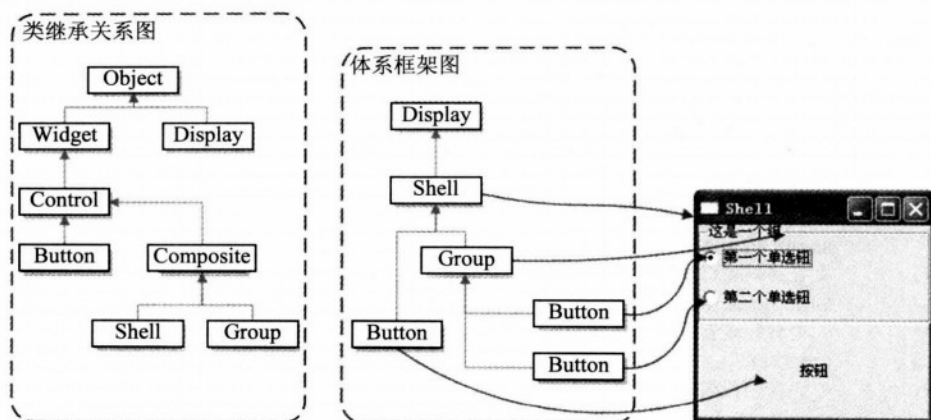


图 5.5 部件与类结构对照图

5.2 按钮 (Button)

按钮是用户最熟悉的部件之一，使用 `Button` 类创建一个按钮实例。按钮有普通按钮 (`SWT.PUSH`)、单选按钮 (`SWT.RADIO`)、多选按钮 (`SWT.CHECK`)、箭头按钮 (`SWT.ARROW`) 和切换按钮 (`SWT.TOGGLE`) 几种类型。

同时，也可以设置按钮的样式。设置按钮文字对齐方式的样式有 `SWT.LEFT`、`SWT.RIGHT` 和 `SWT.CENTER`。设置按钮外观风格的样式有 `SWT.FLAT` 和 `SWT.BORDER`。

5.2.1 普通按钮 (SWT.PUSH)

普通按钮的创建方法如下：

```
Button bt = new Button ( shell , SWT.PUSH);
```

或者：

```
Button bt = new Button ( shell , SWT.NONE);
```

注意：默认的按钮样式为 `SWT.PUSH`。

图 5.6 显示了不同样式组合的普通按钮。

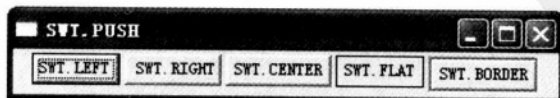


图 5.6 普通按钮

实现的代码如下：

PushButton.java


```
Button bt1 = new Button ( shell , SWT.PUSH|SWT.LEFT);
//设置文本
bt1.setText("SWT.LEFT");
//设置悬浮提示
bt1.setToolTipText("SWT.LEFT");

Button bt2 = new Button ( shell , SWT.PUSH|SWT.RIGHT);
bt2.setText("SWT.RIGHT");
bt2.setToolTipText("SWT.RIGHT");

Button bt3 = new Button ( shell , SWT.PUSH|SWT.CENTER);
bt3.setText("SWT.CENTER");
bt3.setToolTipText("SWT.CENTER");

Button bt4 = new Button ( shell , SWT.PUSH|SWT.FLAT);
bt4.setText("SWT.FLAT");
bt4.setToolTipText("SWT.FLAT");

Button bt5 = new Button ( shell , SWT.PUSH|SWT.BORDER);
bt5.setText("SWT.BORDER");
bt5.setToolTipText("SWT.BORDER");
```

 注意：该代码只显示了程序的核心部分，其他部分与之前例子的代码类似，请读者参阅光盘所附源代码。

5.2.2 切换按钮（SWT.TOGGLE）

切换按钮与普通按钮类似，但是当单击后保持按下状态，再次单击时恢复初始状态。如图 5.7 所示显示的为 3 种不同风格的切换按钮按下时的状态。



图 5.7 切换按钮的不同样式

实现的代码如下：

ToggleButton.java

```
Button bt1 = new Button ( shell , SWT.TOGGLE|SWT.LEFT);
bt1.setText("SWT.LEFT");
bt1.setToolTipText("SWT.LEFT");

Button bt2 = new Button ( shell , SWT.TOGGLE|SWT.FLAT);
bt2.setText("SWT.FLAT");
bt2.setToolTipText("SWT.FLAT");
```



```
Button bt3 = new Button ( shell , SWT.TOGGLE|SWT.BORDER);
bt3.setText("SWT.BORDER");
bt3.setToolTipText("SWT.BORDER");
```

5.2.3 箭头按钮 (SWT.ARROW)

箭头按钮是带有小箭头的按钮，创建时还要指定箭头的方向 SWT.UP、SWT.DOWN、SWT.LEFT 和 SWT.RIGHT。若不指定方向，默认为 SWT.UP。如图 5.8 所示为不同方向的箭头按钮。

实现的代码如下：

ArrowButton.java

```
Button bt1 = new Button ( shell , SWT.ARROW|SWT.UP);
bt1.setToolTipText("SWT.LEFT");

Button bt2 = new Button ( shell , SWT.ARROW|SWT.DOWN);
bt2.setToolTipText("SWT.RIGHT");

Button bt3 = new Button ( shell , SWT.ARROW|SWT.LEFT);
bt3.setToolTipText("SWT.CENTER");

Button bt4 = new Button ( shell , SWT.ARROW|SWT.RIGHT);
bt4.setToolTipText("SWT.FLAT");

Button bt5 = new Button ( shell , SWT.ARROW|SWT.FLAT);
bt5.setToolTipText("SWT.BORDER");

Button bt6 = new Button ( shell , SWT.ARROW|SWT.BORDER);
bt6.setToolTipText("SWT.BORDER");
```



图 5.8 箭头按钮的不同样式

5.2.4 单选按钮 (SWT.RADIO)

单选按钮是在几个按钮中只可以选择一个按钮。如图 5.9 所示为两组不同风格的单选按钮。



图 5.9 不同风格的单选按钮

实现的代码如下：

RadioButton.java

```
//这是第一组单选按钮
Group group1 = new Group(shell, SWT.SHADOW_ETCHED_OUT);
group1.setLayout(new FillLayout(SWT.VERTICAL));
group1.setText("这是一组样式");

Button bt1 = new Button ( group1 , SWT.RADIO|SWT.LEFT);
bt1.setText("SWT.LEFT");
bt1.setToolTipText("SWT.LEFT");

Button bt2 = new Button ( group1 , SWT.RADIO|SWT.RIGHT);
bt2.setText("SWT.RIGHT");
bt2.setToolTipText("SWT.RIGHT");

Button bt3 = new Button ( group1 , SWT.RADIO|SWT.CENTER);
bt3.setText("SWT.CENTER");
bt3.setToolTipText("SWT.CENTER");

//这是第二组单选按钮
Group group2 = new Group(shell, SWT.SHADOW_ETCHED_OUT);
group2.setLayout(new FillLayout(SWT.VERTICAL));
group2.setText("这是另一组样式");

Button bt4 = new Button ( group2 , SWT.RADIO|SWT.FLAT);
bt4.setText("SWT.FLAT");
bt4.setToolTipText("SWT.FLAT");
//设置选中状态
bt4.setSelection( true );

Button bt5 = new Button ( group2 , SWT.RADIO|SWT.BORDER);
bt5.setText("SWT.BORDER");
bt5.setToolTipText("SWT.BORDER");

Button bt6 = new Button ( group2 , SWT.RADIO);
bt6.setText("SWT.RADIO");
bt6.setToolTipText("SWT.RADIO");
```

对于单选按钮，创建对象时要指定按钮所属的父类。也就是说，要说明哪几个按钮属于同一个组。例如，`Button bt = new Button (group , SWT.RADIO)`，这里的 `group` 对象即为单选按钮所属的组。

设定某一个单选按钮为选中状态的方法是 `bt.setSelection(true)`；判断一个按钮是否被选中的方法是 `bt.getSelection()`，如果为 `true` 表示已选中，为 `false` 表示未选中。

5.2.5 多选按钮 (SWT.CHECK)

多选按钮是可以同时选择多个选项的按钮。如图 5.10 所示为几组不同风格的多选按钮。



图 5.10 不同样式的多选按钮

实现的代码如下：

CheckButton.java

```
Button bt1 = new Button ( group , SWT.CHECK|SWT.LEFT);
bt1.setText("SWT.LEFT");
bt1.setToolTipText("SWT.LEFT");

Button bt2 = new Button ( group , SWT.CHECK|SWT.RIGHT);
bt2.setText("SWT.RIGHT");
bt2.setToolTipText("SWT.RIGHT");

Button bt3 = new Button ( group , SWT.CHECK|SWT.CENTER);
bt3.setText("SWT.CENTER");
bt3.setToolTipText("SWT.CENTER");

Button bt4 = new Button ( group , SWT.CHECK|SWT.FLAT);
bt4.setText("SWT.FLAT");
bt4.setToolTipText("SWT.FLAT");

Button bt5 = new Button ( group , SWT.CHECK|SWT.BORDER);
bt5.setText("SWT.BORDER");
bt5.setToolTipText("SWT.BORDER");
```

在设计程序的过程中，通常要获得所选中多选按钮的值。所以为了能方便地获取到哪个按钮被选中了，哪个按钮未被选中，通常把多选按钮创建成数组的形式。例如，`Button[] button = new Button[5]`。之后就可以利用循环来判断是否被选中了。

下面用一个例子说明，有一组单选按钮，单击“确定”按钮后，显示出所选择的所有选项。程序运行后如图 5.11 和图 5.12 所示。

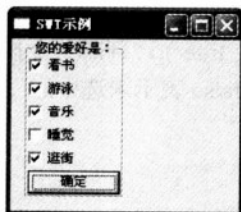


图 5.11 程序运行示例图

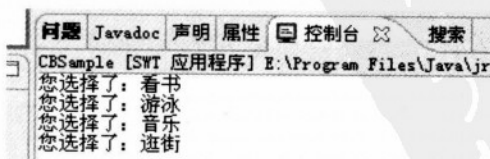


图 5.12 单击“确定”按钮后的结果

以下是该程序的代码：

CBSample.java

```
//定义 Shell 的布局
RowLayout layout = new RowLayout(SWT.VERTICAL);
layout.marginWidth=10;
shell.setLayout(layout);

Group group = new Group(shell, SWT.SHADOW_ETCHED_OUT);
group.setLayout(new FillLayout(SWT.VERTICAL));
group.setText("您的爱好是: ");

//定义按钮为数组，要在事件处理类中使用按钮的引用，要设置成 final 型
final Button[] button = new Button[5];
String[] items = {"看书", "游泳", "音乐", "睡觉", "逛街"};
//初始化按钮数组
for ( int i=0;i<button.length;i++)
{
    button[i] = new Button ( group , SWT.CHECK);
    button[i].setText( items[i] );
}
//确定按钮
Button ok = new Button( group ,SWT.PUSH);
ok.setText("确定");
//添加"确定"按钮事件
ok.addSelectionListener( new SelectionAdapter(){

    public void widgetSelected(SelectionEvent e) {
        //循环所有按钮
        for ( int i=0;i<button.length;i++)
            //如果选中，则输出选中的值
            if ( button[i].getSelection())
                System.out.println("您选择了: "+button[i].getText());
    }

});
```

该程序的实现涉及一些布局和事件处理的方法。有关这两方面的知识将会在下文中详细讲述。

5.2.6 常用的方法

通过以上示例程序，对 Button 类常用的方法进行总结。

- ❑ 设置和获取按钮文本的方法：setText(String string)和 getText()。
 - ❑ 设置和获取按钮是否被选中的方法：setSelection (boolean selected)和 getSelection ()。
- 例如设置某个按钮被选中：button.setSelection (true)。

- ❑ 设置和获取按钮图标的方法：setImage (Image image)和 getImage ()。例如，设置某个按钮图标：button.setImage(display.getImage(SWT.ICON_ERROR));，这里使用的是系统图标。button.setImage(new Image(display, "F:\\icon\\edit.gif"));，这里使用的是保存在 F:\\icon\\edit.gif 路径下的图标文件。
- ❑ 设置和获取按钮文字的对齐方式：setAlignment (int alignment)和 getAlignment ()。例如，设置按钮文字的向右对齐：button.setAlignment(SWT.RIGHT);。另外，可选的对齐方式常量有 SWT.LEFT 和 SWT.CENTER。

5.3 标签 (Label)

SWT 的标签有两类，一类是显示普通文本或图片的标签，一类是分割线。如图 5.13 所示为两种类型的标签示意图。

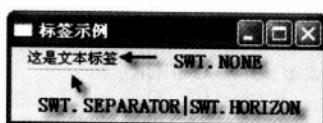


图 5.13 标签示意图

5.3.1 文本标签

创建一个文本标签的代码如下：

```
Label label = new Label ( shell , SWT.NONE);  
label.setText( "这是文本标签");
```

此时使用 SWT.NONE，默认的对齐方式是靠左，若改变对齐方式可以使用 SWT.LEFT、SWT.CENTER 和 SWT.RIGHT。例如，创建一个靠右对齐的标签的代码是 Label label = new Label (shell , SWT.RIGHT);。

5.3.2 分割线标签

创建一个分割线标签的代码如下：

```
Label l = new Label( shell ,SWT.SEPARATOR|SWT.HORIZONTAL|SWT.SHADOW_OUT);
```

创建分割线标签要指定样式类型为 SWT.SEPARATOR。若不指定是垂直还是水平分割线，则默认为垂直方式 (SWT.VERTICAL)。也可设置水平分割线 (SWT. HORIZONTAL)。另外，对分割线还可以选择 3 种不同的外观样式：SWT.SHADOW_IN、SWT.SHADOW_OUT 和 SWT.SHADOW_NONE。

⚠注意：对于分割线标签，若使用 `setText (String string)`方法也不会显示文字。对于设置图标 `setImage (Image image)`也是如此，如果设置了图标，也将不会显示文字。

如图 5.14 所示为标签的各种样式。

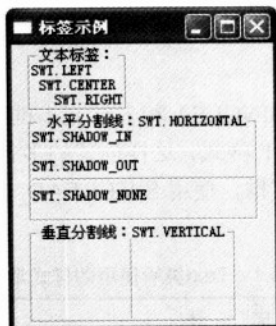


图 5.14 标签的所有样式

⚠注意：该程序的代码在 `LabelSample.java` 文件中，请读者查阅光盘的源代码部分。这里读者只需要了解不同的标签样式即可。

5.3.3 自定义标签 (CLabel)

`Label` 标签类不能将图标和文字同时显示。若要想使图标和文字同时显示要使用 `org.eclipse.swt.custom.CLabel` 类。例如，创建一个带有图标和文字的标签，代码如下：

```
CLabel cl = new CLabel ( shell , SWT.LEFT);
cl.setText( "这是一个带图标的自定义标签");
cl.setImage( display.getSystemImage( SWT.ICON_INFORMATION));
```

代码运行后，显示带有图标的标签，如图 5.15 所示。

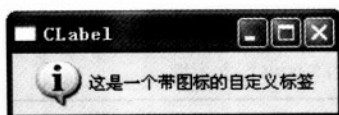



图 5.15 自定义标签 CLabel

5.4 文本框 (Text)

文本框也是常见的控件之一，是可以输入文字的控件。创建一个文本框的代码如下：

```
Text text = new Text (shell, SWT.NONE);
```

 注意: org.eclipse.swt.custom.StyledText 类也是一种文本框, 它的功能更强大。另外, JFace 里有专门针对文本的操作, 它们是功能强大的类, 这些类的使用会在以后的章节详细讲述。

5.4.1 文本框的样式

文本框有单行文本框 (SWT.SINGLE) 和多行文本框 (SWT.MULTI) 两种。如果使用多行文本框, 可以使用 SWT.WRAP 样式, 这样创建的多行文本框当一行填充满后会自动换行。另外, 也可以设置只读的文本框, 使用 SWT.READ_ONLY 样式。表 5.1 显示了创建文本框的 SWT 常量所代表的样式。

表5.1 Text类所使用的样式常量

样 式 常 量	描 述	效 果 图
SWT.SINGLE	单行文本框, 如果不指定 SWT.SINGLE 或 SWT.MULTI, 默认为单行文本框	
SWT.NONE	没有边框的文本框	
SWT.BORDER	带有边框的文本框	
SWT.LEFT	文本框的字符靠左对齐, 默认样式	
SWT.CENTER	文本框的字符居中对齐	
SWT.RIGHT	文本框的字符靠右对齐	
SWT.READ_ONLY	只读文本框	
SWT.PASSWORD	密码输入框	
SWT.MULTI	可输入多行文本的文本框	
SWT.WRAP	多行文本框, 并且自动换行	
SWT.H_SCROLL	带有水平滚动条的多行文本框	
SWT.V_SCROLL	带有垂直滚动条的多行文本框	

5.4.2 文本框程序示例

下面通过一个小程序来说明一个文本框常用的方法。如图 5.16 所示为程序运行后的示意图, 该程序类似编辑器的功能, 可以对文本进行选择, 并且使用剪贴板的功能来实现复制或粘贴功能。



图 5.16 简单编辑器

该程序的具体实现代码如下:

TextSample.java

```
final Text content = new Text ( shell ,SWT.WRAP|SWT.V_SCROLL);
content.setBounds(5,5,325,200);
```

```
Button selectAll = new Button ( shell , SWT.NONE);
selectAll.setText("全选");
selectAll.setBounds( 5,225,75, 25);
selectAll.addSelectionListener( new SelectionAdapter(){
    public void widgetSelected(SelectionEvent e) {
        //选中所有文本
        content.selectAll();
    }
});
```

```
Button cancel = new Button( shell , SWT.NONE);
cancel.setText("取消全选");
cancel.setBounds( 90,225,75, 25);
cancel.addSelectionListener( new SelectionAdapter(){
    public void widgetSelected(SelectionEvent e) {
        //如果有选中的文本
        if(content.getSelectionCount( ) > 0)
            content.clearSelection();//清除选择
    }
});
```

```
Button copy = new Button ( shell , SWT.NONE);
copy.setText("复制");
copy.setBounds( 175,225,75, 25);
copy.addSelectionListener( new SelectionAdapter(){
    public void widgetSelected(SelectionEvent e) {
```



```
//复制到剪贴板
content.copy();
}

});

Button paste = new Button ( shell ,SWT.NONE);
paste.setText("粘贴");
paste.setBounds( 260 ,225,75, 25);
paste.addSelectionListener( new SelectionAdapter(){
    public void widgetSelected(SelectionEvent e) {
        //将剪贴板的内容粘贴
        content.paste();
    }
});
```

该程序主要使用了一些选择文本操作的方法，例如 `selectAll()` 选中所有文本、`clearSelection()` 取消选择等。

5.4.3 常用的方法

除了上例中所使用的方法外，`Text` 类还有许多其他常用的方法，下面来总结一下。

1. 有关文本的方法

- ☐ 设置文本长度的方法：`setTextLimit(int limit)`，参数为文本的最大长度。
- ☐ 设置文本是否可编辑：`setEditable(boolean editable)`，如果为 `false`，则为不可编辑。
- ☐ 设置输入文字的方向：`setOrientation(int orientation)`，如果设置为 `SWT.RIGHT_TO_LEFT`，则输入时从右向左填充。默认是 `SWT.LEFT_TO_RIGHT`。
- ☐ 设置文本输入字符的格式：`setEchoChar(char echo)`，例如，设置类似密码框格式的代码为 `setEchoChar('*')`。
- ☐ 设置输入 Tab 键时退格的字符数：`setTabs(int tabs)`。
- ☐ 向文本中插入字符串的方法：`append(String string)`。
- ☐ 获得文本框内字符串的长度：`getCharCount()`。

2. 有关选择文本操作的方法

- ☐ 选中所有的字符：`selectAll()`。
- ☐ 选中指定字符：`setSelection(int start)`、`setSelection(int start, int end)`和 `setSelection(Point selection)`。例如，`setSelection(5)`，选中从第 5 个字符串开始到最后的字符串。`setSelection(5,10)`选中第 5 个字符到第 10 个字符之间的字符串。`setSelection(new Point(5,10))`与 `setSelection(5,10)`意义相同。
- ☐ 显示所设置的选取文本：`showSelection()`。
- ☐ 取消所有选择：`clearSelection()`。

- ❑ 取得所选中文本的开始位置和结束位置: `getSelection()`, 返回类型为 `Point`。
- ❑ 取得所选取的字符串长度: `getSelectionCount()`。
- ❑ 取得所选取的字符串, 返回类型为 `String`: `getSelectionText()`。
- ❑ 将选取的字符串复制到剪贴板: `copy()`。
- ❑ 将选取的字符串剪切到剪贴板: `cut()`。
- ❑ 将剪贴板上的字符粘贴到文本框: `paste()`。

5.5 列表框 (List)

列表框也是常用控件。如图 5.17 所示的就是一个可以选择多个选项, 并且带有垂直滚动条的列表框。创建一个列表框的代码如下:

```
List list = new List ( shell , SWT.MULTI|SWT.V_SCROLL);
for ( int i=0;i<10;i++)
    list.add("ITEM"+i);
```



图 5.17 列表控件

向列表框中添加选项可使用 `add(String string)` 方法, `List` 类还提供了 `setItems(String[] items)` 将字符串数组的值添加到列表框中。例如, 上面的代码可修改为如下所示, 效果是一样的。

```
List list = new List ( shell , SWT.MULTI|SWT.V_SCROLL);
String[] items = new String[10];
for ( int i=0;i<items.length;i++)
    items[i]="ITEM"+i;
list.setItems( items );
```


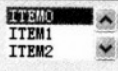
5.5.1 列表框的样式

列表框的样式很简单, 如表 5.2 所示为显示创建列表框的 SWT 常量所代表的样式。

表5.2 List类所使用的样式常量

样 式 常 量	描 述	效 果 图
SWT.SINGLE	只能选中一个选项, 为默认样式	
SWT.MULTI	可同时选择多个选项的列表框, 按住 Ctrl 键即可多选	

续表

样 式 常 量	描 述	效 果 图
SWT.H_SCROLL	带有水平滚动条的多行列表框	
SWT.V_SCROLL	带有垂直滚动条的多行列表框	

5.5.2 列表框程序示例

下面以一个小程序来说明列表框的一些常用方法。如图 5.18 所示为该程序运行后的效果图。该程序的功能是，可以将左侧列表框中的值移动到右侧的列表框中，也可以将右侧列表框中的值移到左侧列表框中。单击“>”按钮，只对左侧选中的选项移动到右侧；单击“>>”按钮将全部左侧的选项移动到右侧的列表框中。同理“<”和“<<”与之类似。另外，对右侧列表框中的选项还可以进行排序。单击“上”按钮，将所选的选项向上移动一个位置。同理，单击“下”按钮，将所选的选项向下移动一个位置。

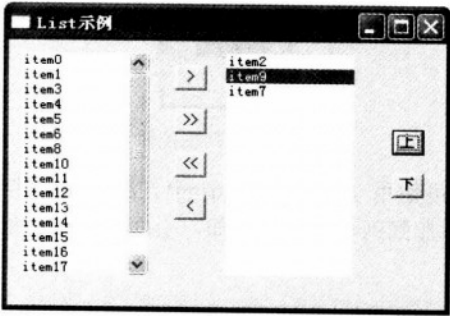


图 5.18 程序运行后效果图

该程序的具体实现代码如下：

ListSample.java

```
package com.fengmanfei.ch5;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.SelectionAdapter;
import org.eclipse.swt.events.SelectionEvent;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.List;
import org.eclipse.swt.widgets.Shell;

public class ListSample {

    public static void main(String[] args) {
```

```

Display display = new Display();
Shell shell = new Shell(display);
shell.setText("List 示例");

String[] itemLeft = new String[20]; // 定义保存左侧列表框中的数据
String[] itemRight = new String[0]; // 定义保存右侧列表框中的数据
// 初始化左侧字符串数组
for (int i = 0; i < 20; i++)
    itemLeft[i] = "item" + i;

// 定义左侧列表框, 可选择多个并且带有垂直滚动条
final List left = new List(shell, SWT.MULTI | SWT.V_SCROLL);
left.setBounds(10, 10, 100, 180); // 设置位置和大小
left.setItems(itemLeft); // 设置选项数据
left.setToolTipText("请选择列表项"); // 设置提示

// 定义右侧列表框, 可选择多个并且带有垂直滚动条
final List right = new List(shell, SWT.MULTI | SWT.V_SCROLL);
right.setBounds(170, 10, 100, 180);
right.setItems(itemRight);
right.setToolTipText("已选择的列表项");

// 创建事件监听类, 为内部类
SelectionAdapter listener = new SelectionAdapter() {
    // 按钮单击事件的处理方法
    public void widgetSelected(SelectionEvent e) {
        // 取得触发事件的控件对象
        Button b = (Button) e.widget;
        if (b.getText().equals(">")) // 如果是">"按钮
            verifyValue(left.getSelection(), left, right);
        else if (b.getText().equals(">>")) // 如果是">>"按钮
            verifyValue(left.getItems(), left, right);
        else if (b.getText().equals("<")) // 如果是"<"按钮
            verifyValue(right.getSelection(), right, left);
        else if (b.getText().equals("<<")) // 如果是"<<"按钮
            verifyValue(right.getItems(), right, left);
        else if (b.getText().equals("上")) // 如果是"上"按钮
        {
            // 获得当前选中选项的索引值
            int index = right.getSelectionIndex();
            if (index <= 0) // 如果没有选中, 则返回
                return;
            // 如果选中了选项值, 获得当前选项的值
            String currentValue = right.getItem(index);
            // 将选中的选项与上一个选项交换值
            right.setItem(index, right.getItem(index - 1));
            right.setItem(index - 1, currentValue);
            // 设定上一个选项为选中状态
            right.setSelection(index - 1);
        }
    }
};

```

```

    } else if (b.getText().equals("下"))// 如果是"下"按钮
    {
        // 与上移按钮的逻辑相同
        int index = right.getSelectionIndex();
        if (index >= right.getItemCount() - 1)
            return;
        String currentValue = right.getItem(index);
        right.setItem(index, right.getItem(index + 1));
        right.setItem(index + 1, currentValue);
        right.setSelection(index + 1);
    }
}

//改变左右列表值
public void verifyValue(String[] select, List from, List to) {
    // 循环所有选中的选项值
    for (int i = 0; i < select.length; i++) {
        // 从一个列表中移出该选项值
        from.remove(select[i]);
        // 添加到另一个列表中
        to.add(select[i]);
    }
}

};

// 定义右移按钮
Button bt1 = new Button(shell, SWT.NONE);
bt1.setText(">");
bt1.setBounds(130, 20, 25, 20);
// 并为按钮注册事件, 其他的按钮类似
bt1.addSelectionListener(listener);

Button bt2 = new Button(shell, SWT.NONE);
bt2.setText(">>");
bt2.setBounds(130, 55, 25, 20);
bt2.addSelectionListener(listener);

Button bt3 = new Button(shell, SWT.NONE);
bt3.setText("<<");
bt3.setBounds(130, 90, 25, 20);
bt3.addSelectionListener(listener);

Button bt4 = new Button(shell, SWT.NONE);
bt4.setText("<");
bt4.setBounds(130, 125, 25, 20);
bt4.addSelectionListener(listener);

Button bt5 = new Button(shell, SWT.NONE);

```



```
bt5.setText("上");
bt5.setBounds(300, 70, 25, 20);
bt5.addSelectionListener(listener);

Button bt6 = new Button(shell, SWT.NONE);
bt6.setText("下");
bt6.setBounds(300, 105, 25, 20);
bt6.addSelectionListener(listener);

shell.setSize(350, 250);
shell.open();
while (!shell.isDisposed()) {
    if (!display.readAndDispatch())
        display.sleep();
}
display.dispose();
}
```

该程序主要设置了对选项值的一些读取操作，而且还涉及了使用内部类来处理事件的方法，如下代码：

```
SelectionAdapter listener = new SelectionAdapter(){...}
```

使用这种方式处理事件的好处是可以在一个类中对不同的事件集中处理。如果这里读者有疑问也没有关系，将会在下文中对事件的处理进行详细的讲述。

5.5.3 常用的方法

列表框中的方法主要是对选项进行操作，下面就来总结一下常用的方法。

1. 向列表框中添加选项

- ☐ 向列表框的末尾添加选项：add(String string)。
- ☐ 向列表指定位置添加选项：add(String string, int index)和 setItem(int index, String string)。
- ☐ 将字符串数组添加到列表框：setItems(String[] items)。

2. 删除列表框中的选项

- ☐ 删除指定一个索引出的选项：remove(int index)。
- ☐ 删除多个索引的选项：remove(int[] indices)。
- ☐ 删除指定开始和结束索引值间的选项：remove(int start, int end)。
- ☐ 删除指定字符串的选项：remove(String string)。
- ☐ 删除列表中所有的选项：removeAll()。

3. 获得列表框的状态

- ❑ 获得指定索引的选项值，返回 String 类型：getItem(int index)。
- ❑ 获得选项的总数：getItemCount()。
- ❑ 获得所有的选项值，返回 String[] 数组：getItems()。
- ❑ 获得第一个选项的索引值：getTopIndex()。
- ❑ 查找是否存在指定的选项，如果存在，则返回所在索引值：indexOf(String string)。
- ❑ 查找指定开始索引后是否存在指定的选项，如果存在，则返回索引值：indexOf(String string, int start)。

4. 对列表框中已选中的选项操作方法

- ❑ 获得当前所选中的所有选项，返回 String[] 数组：getSelection()。
- ❑ 获得当前选中的选项数目：getSelectionCount()。
- ❑ 获得当前选中选项的索引值：getSelectionIndex()。
- ❑ 获得当前选中选项的索引值数组，返回 int[] 数组：getSelectionIndices()。
- ❑ 判断某个索引值的选项是否已被选中：isSelected(int index)，若为 true，则已被选中，为 false，则未被选中。
- ❑ 设置指定索引值的选项为选中状态：select(int index)和 setSelection(int index)。
- ❑ 设置指定索引值数组的选项为选中状态：select(int[] indices)和 setSelection(int[] indices)。
- ❑ 设置开始索引到结束索引的选项为选中状态：select(int start, int end)和 setSelection(int start, int end)。
- ❑ 全选：selectAll()。
- ❑ 设置指定字符串数组中值为选中状态：setSelection(String[] items)。
- ❑ 显示选中的方法：showSelection()。
- ❑ 反选指定索引的选项：deselect(int index)。
- ❑ 反选指定索引数组的选项：deselect(int[] indices)。
- ❑ 反选开始索引到结束索引的选项：deselect(int start, int end)。
- ❑ 全部反选：deselectAll()。

5.6 组合框 (Combo)

组合框与列表框类似，也是由多个选项构成的。如图 5.19 所示为一个典型的组合框。创建一个组合框的代码如下：

```
Combo combo = new Combo ( shell , SWT.DROP_DOWN);
String[] items = new String[4];
for ( int i=0;i<items.length;i++)
    items[i]="ITEM"+i;
combo.setItems( items );
```

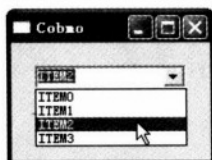



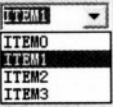
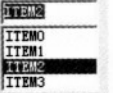
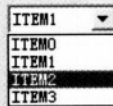
图 5.19 组合框示意图

由于组合框与列表框的功能差不多，只是二者在显示的外观上不太一样，所以对组合框的方法就不再介绍了，请读者参阅列表框一节中的介绍。

5.6.1 组合框的样式

如表 5.3 所示为创建组合框的 SWT 常量所代表的样式。

表5.3 Combo类所使用的样式常量

样 式 常 量	描 述	效 果 图
SWT.DROP_DOWN	以下拉的形式显示选项，为默认样式	
SWT.SIMPLE	选项以列表的形式在下方显示	
SWT.READ_ONLY	编辑框的值不可以修改	

5.6.2 组合框程序示例

下面以一个小程序说明组合框事件的一些方法。如图 5.20 所示为一个简单的组合框。默认情况下，单击键盘上的上下箭头按钮可以对选项进行上下选择，但左右按钮不起作用。该示例的程序使左右按钮也可以进行上下选择，与默认的上下箭头的作用一样。

该程序的代码如下：

ComboSample.java

```
package com.fengmanfei.ch5;

import org.eclipse.swt.SWT;
import org.eclipse.swt.events.KeyAdapter;
import org.eclipse.swt.events.KeyEvent;
```

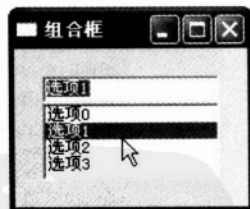


图 5.20 组合框小程序

```
import org.eclipse.swt.layout.FillLayout;
import org.eclipse.swt.widgets.Combo;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;

public class ComboSample {

    public static void main(String[] args) {
        Display display = new Display();
        Shell shell = new Shell(display);
        shell.setText("组合框");
        FillLayout layout = new FillLayout();
        layout.marginHeight = 20;
        layout.marginWidth = 20;
        shell.setLayout(layout);

        final Combo combo = new Combo(shell, SWT.SIMPLE);
        String[] items = new String[4];
        for (int i = 0; i < items.length; i++)
            items[i] = "选项" + i;
        combo.setItems(items);
        // 注册键盘事件
        combo.addListener(new KeyAdapter() {
            public void keyPressed(KeyEvent e) {
                // 如果单击了向左的箭头按键, 则选中上一个选项
                if (e.keyCode == SWT.ARROW_LEFT)
                    combo.select(combo.getSelectionIndex() - 1);
                // 如果单击了向右的箭头按键, 则选中下一个选项
                else if (e.keyCode == SWT.ARROW_RIGHT)
                    combo.select(combo.getSelectionIndex() + 1);
            }
        });
        shell.setSize(200, 100);
        shell.pack();
        shell.open();
        while (!shell.isDisposed()) {
            if (!display.readAndDispatch())
                display.sleep();
        }
        display.dispose();
    }
}
```

通过这段程序, 又接触到了一个新的事件类型, 即键盘事件。注册键盘事件的方法是 `addListener(KeyListener listener)`。随着不断地深入, 读者将会接触到更多不同的事件类型。

5.6.3 组合框的常用方法

组合框的大部分方法与列表框的方法类似，也有一些与其不同的方法。

1. 组合框中对剪贴板的操作方法

- ❑ 复制：copy()。
- ❑ 剪切：cut()。
- ❑ 粘贴：paste()。

2. 列表框中所没有的方法

- ❑ 清除文本框中的字符：clearSelection()。
- ❑ 获得文本框中的字符，返回类型为 String：getText()。
- ❑ 设置组合框文字输入的方向：setOrientation(int orientation)，例如，设置由右向左输入的代码为 combo.setOrientation(SWT.RIGHT_TO_LEFT);，默认为 SWT. LEFT_TO_RIGHT。
- ❑ 设置文本框最大字符数：setTextLimit(int limit)。

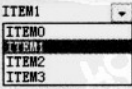
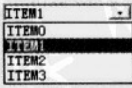
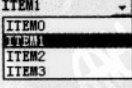
5.6.4 自定义组合框 CCombo 类

除了使用 Combo 类以外，org.eclipse.swt.custom.CCombo 类也具有组合框的功能，而且相对 Combo 增加了不同的样式。创建自定义组合框的代码如下：

```
CCombo combo = new CCombo ( shell , SWT.FLAT);
String[] items = new String[4];
for ( int i=0;i<items.length;i++)
    items[i]="ITEM"+i;
combo.setItems( items );
```

如表 5.4 所示为创建自定义组合框的 SWT 常量所代表的样式。

表5.4 CCombo类所使用的样式常量

样 式 常 量	描 述	效 果 图
SWT. FLAT	具有平面效果的组合框	
SWT. BORDER	具有立体边框的组合框	
SWT. READ_ONLY	只读状态的组合框	

🔔注意: CCombo 类 SWT.SIMPLE 常量, 只使用带下拉按钮的组合框。

5.7 本章小结

通过本章的学习, 读者应该对几种常用的控件有了一个比较清晰的认识, 例如按钮 (Button)、标签 (Label)、文本框 (Text)、列表框 (List) 和组合框 (Combo) 等。同时, 本章也介绍了两个自定义控件 CLabel 和 CCombo。虽然本章对事件和布局涉及了一些, 但这并不是本章的重点。



第 6 章 面板容器类

本章将详细介绍 SWT 中常见的容器类，首先介绍所有容器类的父类 Composite 类，然后分别介绍常见容器类：分组框(Group)、选项卡(TabFolder)、自定义选项卡(CTabFolder)、分割窗框(SashForm)、自定义分割框(CBanner)、滚动面板(ScrolledComposite)和 ViewForm。

6.1 面板类 (Composite)

容器可以理解为可以放置其他控件的控件，试想一下如果一个窗口上放置了太多的控件，这时就需要将控件分类，然后放到不同的容器中，这样窗口上的控件就整齐多了。

图 6.1 显示了 Composite 类的继承关系图。

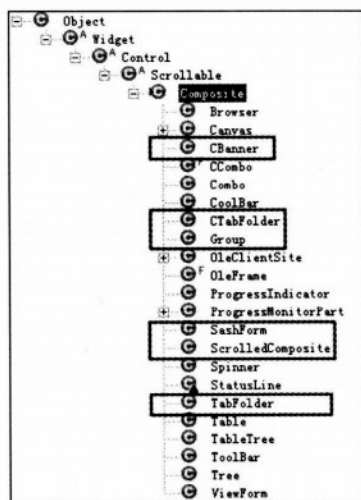


图 6.1 Composite 类继承关系图

注意：图中所画框的面板类是本章中所要介绍的。


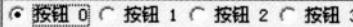

创建一个面板类的代码如下：

```
Composite composite = new Composite(shell, SWT.NONE);
```

6.1.1 面板类的样式

表 6.1 所示为面板类的 SWT 常量所代表的样式。

表6.1 Composite类所使用的样式常量

样 式 常 量	描 述	效 果 图
SWT.NONE	没有边框的面板	
SWT.BORDER	带有边框的面板	
SWT.NO_RADIO_GROUP	对于一组单选按钮，可以同时选中多个按钮，默认情况下只允许选中一个单选按钮	

6.1.2 面板类的常用方法

面板类主要是对布局和面板中放置的控件进行操作，主要方法有：

- ❑ 获得面板中所有控件的方法：Control[] getChildren()。
- ❑ 获得面板的父面板：Composite getParent()。
- ❑ 设置面板布局：setLayout(Layout layout)。
- ❑ 刷新布局：layout()。

6.2 分组框（Group）

分组框是能够显示标题分组的面板类，如图 6.2 所示就是一个分组框。

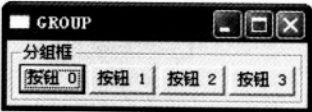


图 6.2 分组框示意图

该程序的具体实现代码如下：

```
//创建分组框
Group group = new Group (shell, SWT.NONE);
group.setText("分组框");//设置标题
group.setLayout (new RowLayout ());//设置布局
//添加按钮
for (int i=0; i<4;i++) {
    Button button = new Button (group, SWT.PUSH);
    button.setText ("按钮 " + i);
}
```

⚠注意：分组框设置标题的方法为 group.setText("分组框") 。

由于分组框的样式根据操作系统的不同有不同的样式，这里只给出可以选择使用的样式常量，具体每种样式的外观要在不同的系统中才会有所不同。

Group 类可以选择的样式常量有：SWT.SHADOW_ETCHED_IN、SWT.SHADOW_

ETCHED_OUT、SWT.SHADOW_IN、SHADOW_OUT 和 SWT.SHADOW_NONE。

6.3 选项卡 (TabFolder)

选项卡是带有标签页的容器，一个选项卡文件夹 (TabFolder) 由一个或多个选项卡 (TabItem) 组成，每个选项卡都可以显示一个控件，并且一次只显示一个选项卡。图 6.3 所示为一个选项卡容器，图 6.4 所示为当窗口缩小时选项卡的效果图。

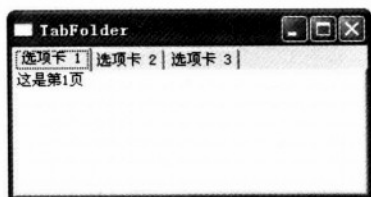


图 6.3 TabFolder 选项卡容器

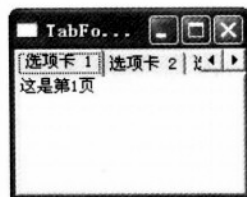


图 6.4 窗口缩小后选项卡的效果图

创建图中所示选项卡的代码如下：

TabFolderSample.java

```
//创建 TabFolder 对象
final TabFolder tabFolder = new TabFolder (shell, SWT.TOP);
//设置选项卡的布局
tabFolder.setLayout( new FillLayout());
for (int i=1; i<4; i++) {
    //创建 TabItem 选项卡标签对象
    TabItem item = new TabItem (tabFolder, SWT.NONE);
    //设置选项卡的文本
    item.setText ("选项卡 " + i);
    Text t = new Text (tabFolder, SWT.MULTI);
    t.setText ("这是第" + i+"页");
    //设置选项卡所控制的控件
    item.setControl (t);
}
tabFolder.pack ();
```

6.3.1 选项卡的基本构成

从以上的代码中可以发现，要使用选项卡需要两个类：TabFolder 和 TabItem。这两个类在 SWT 中的继承关系如图 6.5 和图 6.6 所示。

一个 TabFolder 由一个或多个 TabItem 构成，它们之间是一对多的关系。TabItem 通过 setControl(Control control)方法设置所控制显示的控件。TabItem 控制的控件可以是基本的组件，也可以是面板容器类。图 6.7 显示了 TabFolder、TabItem 与各种控件之间的关系。

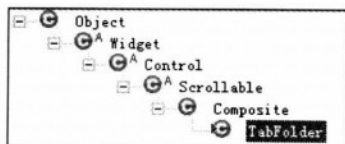


图 6.5 TabFolder 类继承关系图

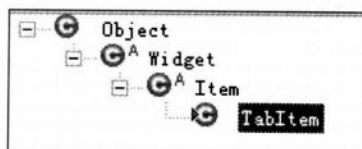


图 6.6 TabItem 类继承关系图

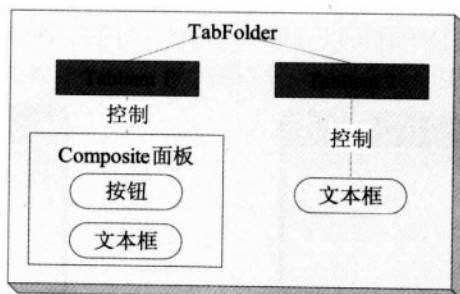


图 6.7 TabFolder、TabItem 与各种控件之间的关系

像这种 TabFolder 与 TabItem 对应的关系，SWT 中还有许多类也是如此，如 Table 和 TableItem，Menu 和 MenuItem 等，也都是一对多的关系，所以理解了它们的关系对以后的学习扫清了障碍。

6.3.2 设置底部显示选项卡

选项卡的位置也可以放在底部，只需要在创建时使用样式 SWT.BOTTOM 即可。例如，要想使上个例子中的选项卡显示在底部，只需创建时使用以下代码：

```
final TabFolder tabFolder = new TabFolder(shell, SWT.BOTTOM);
```

使用底部显示选项卡的效果如图 6.8 所示。

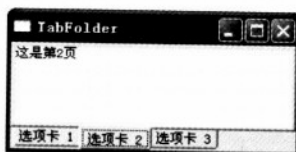


图 6.8 底部显示选项卡

6.3.3 设置选项卡图标

可以为每个选项卡设置图标，这样看起来更美观。为选项卡设置图标的方法很简单，只需使用 TabItem 的 setImage(Image image) 方法。例如，以下是创建选项卡图标的代码：

```
Image image = new Image(display, "F:\\javaDev\\eclipse\\plugins\\org.eclipse.platform_3.1.2\\eclipse.gif");  
item.setImage( image );
```

其中 “F:\javaDev\ eclipse\plugins\org.eclipse.platform_3.1.2\ eclipse.gif” 使用的是 Eclipse 的图标文件的地址。设置图标后的选项卡如图 6.9 所示。



图 6.9 带有图标的选项卡

6.3.4 选项卡的常用方法

在 TabFolder 类中,控制 TabItem 与 List 列表框中的选项的方法差不多,主要有以下内容:

- ❑ 获得指定索引的选项卡: `TabItem getItem(int index)`, `index` 为索引值。
- ❑ 获得选项卡的总数, 返回值为 `int`: `getItemCount()`。
- ❑ 获得选项卡数组: `getItems()`。
- ❑ 获得当前选中的选项卡: `getSelection()`。
- ❑ 获得当前选中选项卡的索引值, 返回值为 `int`: `getSelectionIndex()`。
- ❑ 查找是否存在指定的选项, 返回 `int`: `indexOf(TabItem item)`, 若存在, 返回选项卡所在的索引值, 若不存在, 返回 -1。
- ❑ 设置选中指定选项卡: `setSelection(TabItem[] items)`。
- ❑ 设置选中指定索引值的选项卡: `setSelection(int index)`。

6.4 自定义选项卡 (CTabFolder)

在 `org.eclipse.swt.custom` 包中,自定义选项卡 `CTabFolder` 类比 `TabFolder` 类的功能更强大,自定义选项卡除了样式上更加丰富外,也增加了许多常用方法。同样, `CTabFolder` 由多个 `CTabItem` 对应。图 6.10 显示的是一个自定义选项卡的效果,图 6.11 是当窗口缩小时的样式。

CTabFolderSample.java

```
// 创建 CTabFolder 对象
final CTabFolder tabFolder = new CTabFolder(shell, SWT.TOP);
// 设置选项卡的布局
//tabFolder.setLayout(new FillLayout());
//设置选项标签的高度
```

```

tabFolder.setTabHeight(20);
//设置上下，左右补白
tabFolder.marginHeight=10;
tabFolder.marginWidth=10;
for (int i = 1; i < 4; i++) {
    // 创建 CTabItem 选项卡标签对象
    CTabItem item = new CTabItem(tabFolder, SWT.NONE);
    // 设置选项卡的文本
    item.setText("选项卡 " + i);
    Text t = new Text(tabFolder, SWT.MULTI);
    t.setText("这是第 " + i + " 页");
    // 设置选项卡所控制的控件
    item.setControl(t);
}
tabFolder.pack();

```

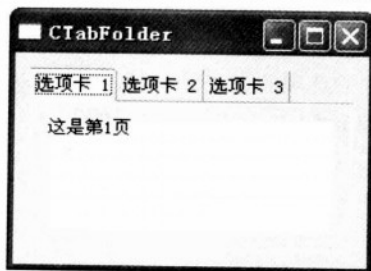


图 6.10 CTabFolder 的效果图

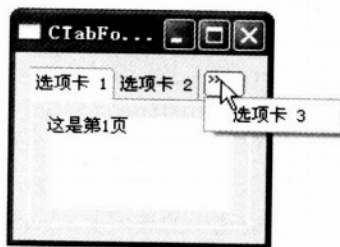


图 6.11 窗口缩小后的效果

6.4.1 带有“关闭”按钮的选项卡

自定义选项卡可以使用带有“关闭”按钮的选项卡，使用这种样式的选项卡只需要创建 CTabFolder 对象时使用 SWT.CLOSE 样式常量即可。例如，修改以上第一行程序的代码为：

```
final CTabFolder tabFolder = new CTabFolder(shell, SWT.CLOSE);
```

程序运行后效果如图 6.12 所示。

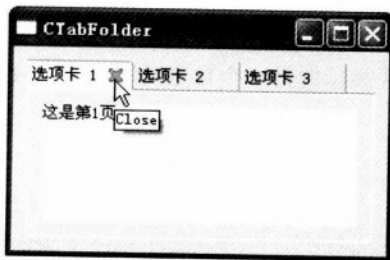


图 6.12 带有“关闭”按钮的选项卡

当单击“关闭”按钮时，关闭选项卡，则其 CTabItem 对象也就销毁了，这个选项中的控件也销毁了。

6.4.2 带有边框的选项卡

如果要想选项卡加上边框,可以使用 `SWT.BORDER` 常量,例如创建一个既带有“关闭”按钮,又带有边框的选项卡,代码如下:

```
final CTabFolder tabFolder = new CTabFolder(shell, SWT.CLOSE|SWT.BORDER);
```

带有边框的选项卡如图 6.13 所示。

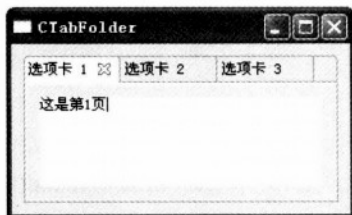


图 6.13 带有边框的选项卡

6.4.3 显示“最大化/最小化”按钮

自定义选项卡还可以显示“最大化/最小化”按钮,设置的方法如以下代码:

```
//显示"最大化"按钮  
tabFolder.setMaximizeVisible(true);  
//显示"最小化"按钮  
tabFolder.setMinimizeVisible(true);
```

显示出“最大化”和“最小化”按钮的选项卡效果如图 6.14 所示。

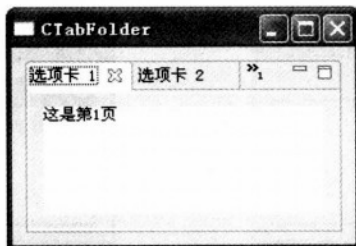


图 6.14 显示“最大化”和“最小化”按钮

虽然显示了最大化和最小化按钮,但如果不作任何事件的处理,单击“最大化”和“最小化”按钮界面不会发生任何变化。在 6.4.5 节的示例中将涉及最大化和最小化处理的问题。

6.4.4 设置选项卡的颜色和背景图片

自定义选项卡界面很友好,可以设置选中状态标签的颜色或图片。

1. 设置单一的前景色和背景色

背景色是整个选项卡背景的颜色，而前景色是选项卡文字的颜色。通过使用 `setSelectionBackground(Color color)` 方法设置背景色，使用 `setSelectionForeground(Color color)` 方法设置前景色。例如将文字颜色设置为白色，背景色设置为蓝色的代码如下：

```
tabFolder.setSelectionForeground( display.getColor( SWT.COLOR_WHITE ));  
tabFolder.setSelectionBackground( display.getColor( SWT.COLOR_BLUE ));
```

设置前景色和背景色后的效果如图 6.15 所示。

`display.getColor(SWT.COLOR_WHITE)` 方法是获得系统颜色，有关颜色的知识会在 SWT 的资源管理中介绍。

2. 设置渐变的背景色

自定义选项卡还可以不使用一种颜色，而是使用多种颜色，通过渐变的方式显示选项卡。创建渐变颜色背景的方法是 `setSelectionBackground(Color[] colors,int[] percents)` 或 `setSelectionBackground(Color[] colors, int[] percents, boolean vertical)`。其中第二个方法中的 `vertical` 如果为 `true`，则表示垂直渐变颜色；如果为 `false`，则表示水平渐变。默认为水平渐变。例如，创建一个渐变背景色的选项卡，代码如下：

```
Color[] color = new Color[4];  
color[0] = display.getColor(SWT.COLOR_DARK_BLUE);  
color[1] = display.getColor(SWT.COLOR_BLUE);  
color[2] = display.getColor(SWT.COLOR_WHITE);  
color[3] = display.getColor(SWT.COLOR_WHITE);  
int[] intArray = new int[] {25, 50, 100};  
//设置渐变颜色  
tabFolder.setSelectionBackground(color,intArray);
```

渐变颜色显示的选项卡如图 6.16 所示。

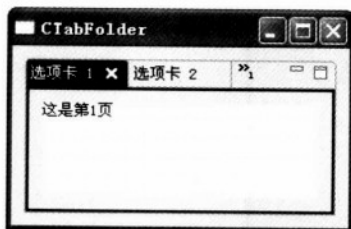


图 6.15 设置背景色和前景色

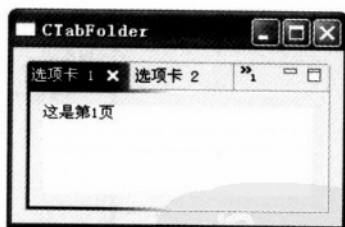


图 6.16 设置了渐变色的选项卡

3. 设置选项卡背景图片

创建选项卡背景图片的方法是 `setSelectionBackground(Image image)`。例如，设置一个图片的选项卡的代码如下：

```
tabFolder.setSelectionBackground( new Image(display,"F:\\javaDev\\eclipse\\plugins\\ org.eclipse.  
platform_3.1.2\\eclipse.png"));
```

设置图片后的选项卡效果如图 6.17 所示。

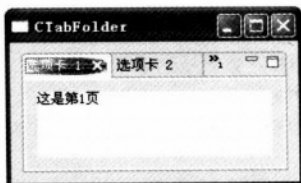


图 6.17 设置背景图片后的选项卡

当然这里的图片是随便使用的一个图片，但对于设计页面来说，可以制作出合适大小的图片来设置选项卡的背景，从而产生不同的效果。

注意：若同时设置背景颜色和背景图片，只显示背景的图片，而设置的颜色不起作用。

6.4.5 仿 Eclipse 编辑区的选项卡

细心的读者可能已经觉察到，自定义选项卡的外观样式与 Eclipse 工作区的选项卡非常类似，如图 6.18 所示为 Eclipse 工作区中的一个选项卡。下面这个完整的例子，完全仿照 Eclipse 编辑区的选项卡来定义选项卡。

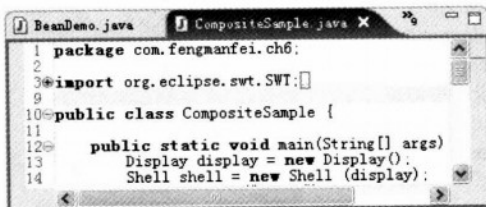


图 6.18 Eclipse 编辑区的选项卡

以下程序为仿照 Eclipse 编辑区选项卡的全部代码：

EclipseTabSample.java

```
package com.fengmanfei.ch6;
import org.eclipse.swt.SWT;
import org.eclipse.swt.custom.CTabFolder;
import org.eclipse.swt.custom.CTabFolder2Adapter;
import org.eclipse.swt.custom.CTabFolderEvent;
import org.eclipse.swt.custom.CTabItem;
import org.eclipse.swt.graphics.Image;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.widgets.Text;

public class EclipseTabSample {
```



```

public static void main(String[] args) {
    Display display = new Display();
    // 创建图片对象, 该图片对象设置选项卡上的图标
    Image image = new Image(display,
"F:\\javaDev\\eclipse\\plugins\\org.eclipse.platform_3.1.2\\intro\\css\\graphics\\icons\\etool\\samples.gif");

    final Shell shell = new Shell(display);
    shell.setText("仿 Eclipse 编辑区的选项卡");
    shell.setLayout(new GridLayout());
    // 创建自定义选项卡对象
    final CTabFolder folder = new CTabFolder(shell, SWT.BORDER);
    // 设置选项卡的布局, 通过布局的设置呈现出最大化 and 最小化的外观
    folder.setLayoutData(new GridData(SWT.FILL, SWT.FILL, true, false));
    // 设置复杂的选项卡, 也就是带有圆角的选项卡标签
    folder.setSimple(false);
    // 设置未选中标签, 图标和"关闭"按钮的状态
    folder.setUnselectedImageVisible(true);
    folder.setUnselectedCloseVisible(true);
    // 设置前景色和背景色
    folder.setSelectionForeground(display.getSystemColor(SWT.COLOR_WHITE));
    folder.setSelectionBackground(display.getSystemColor(SWT.COLOR_BLUE));
    // 显示"最大化"和"最小化"按钮
    folder.setMinimizeVisible(true);
    folder.setMaximizeVisible(true);
    // 创建选项卡标签对象
    for (int i = 1; i < 5; i++) {
        CTabItem item = new CTabItem(folder, SWT.CLOSE);
        item.setText("选项卡 " + i);
        item.setImage(image);
        // 每个选项卡中放置一个 Text 文本框
        Text text = new Text(folder, SWT.MULTI | SWT.V_SCROLL
| SWT.H_SCROLL);
        // 文本框中的文字带有\n 表示, 显示时换到下一行
        text.setText("这是第" + i + "页:\n 该选项卡仿照 Eclipse 设计\n 最大化和最小化按钮都可以使用");
        item.setControl(text);
    }
    // 注册选项卡事件
    folder.addCTabFolder2Listener(new CTabFolder2Adapter() {
        // 当单击"最小化"按钮时触发的事件
        public void minimize(CTabFolderEvent event) {
            // 设置选项卡的状态为最小化, 选项卡的状态决定显示在右上角的窗口按钮
            folder.setMinimized(true);
            // 改变选项卡的布局, 呈现最小化状态
            folder.setLayoutData(new GridData(SWT.FILL, SWT.FILL, true, false));
            // 刷新布局, 否则重新设置的布局将不起作用
            shell.layout(true);
        }
    });
    // 当单击"最大化"按钮时触发的事件

```



```

public void maximize(CTabFolderEvent event) {
    folder.setMaximized(true);
    folder.setLayoutData(new GridData(SWT.FILL, SWT.FILL, true, true));
    shell.layout(true);
}
// 当单击“还原”按钮时触发的事件
public void restore(CTabFolderEvent event) {
    folder.setMinimized(false);
    folder.setMaximized(false);
    folder.setLayoutData(new GridData(SWT.FILL, SWT.FILL, true, false));
    shell.layout(true);
}
});
shell.setSize(300, 200);
shell.open();
while (!shell.isDisposed()) {
    if (!display.readAndDispatch())
        display.sleep();
}
// 释放图片资源
image.dispose();
display.dispose();
}
}

```

该程序运行后，效果如图 6.19 所示，窗口最大化后的效果如图 6.20 所示。

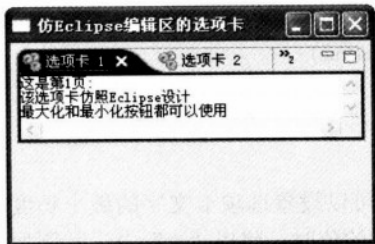


图 6.19 程序运行后效果图



图 6.20 最大化窗口后的效果图

实现该程序主要有以下几个关键点：

1. 选项卡外观的设计

除了之前使用的设置图标、前景色和背景色的有关方法外，该程序还使用了 `folder.setSimple(false)` 方法，这样选项卡的外观是以圆角形式来显示的，看起来更美观。

2. “最大化”和“最小化”按钮事件的处理

光有按钮，单击“最大化”和“最小化”按钮是不起作用的，需要配合相应的事件处理才可以产生响应，注册最大化和最小化事件的代码如下：

```

folder.addCTabFolder2Listener(new CTabFolder2Adapter() {
    public void minimize(CTabFolderEvent event) {

```

```

        // 当单击"最小化"按钮时触发的事件
    }
    public void maximize(CTabFolderEvent event) {
        // 当单击"最大化"按钮时触发的事件
    }
    public void restore(CTabFolderEvent event) {
        // 当单击"还原"按钮时触发的事件
    }
}
});

```

3. 最大化和最小化界面的外观的实现原理

最大化和最小化界面的改变是通过改变布局来实现的,例如,当最大化时,可以设置布局为水平和垂直充满整个窗口,这样就达到了放大的目的。本程序中使用了网格格式(GridLayout)布局。有关布局的知识将在第7章作详细的介绍。

4. 选项卡的有关状态

根据选项卡的状态,会决定显示的是显示在右上角的是“最大化”按钮还是“还原”按钮。获得选项卡状态的方法是:是否最小化 `getMinimized()`,是否最大化 `getMaximized()`。设置选项卡状态的方法是:设置是否最小化 `setMinimized(boolean minimize)`,设置是否最大化 `setMaximized(boolean maximize)`。图 6.21~图 6.23 显示了设置不同状态时按钮的变化。

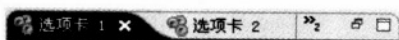


图 6.21 `setMinimized(true)`

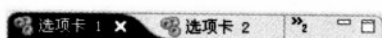


图 6.22 `setMinimized(false)`且 `setMaximized(false)`

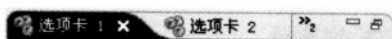


图 6.23 `setMaximized(true)`

6.4.6 限制选项卡文字的长度

通过使用 `setMinimumCharacters(int count)`方法可以设置选项卡文字的最小长度,其中 `count` 是文字的字符长度,当选项卡文字大于 `count` 的值时,将以“...”表示。例如,限制选项卡字符长度为 2 的代码如下:

```
tabFolder.setMinimumCharacters(2);
```

限制字符以后的选项卡效果如图 6.24 所示。

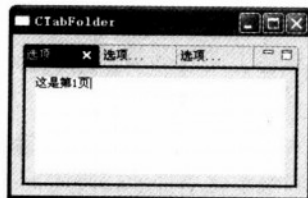


图 6.24 限制了选项卡文字长度的效果

6.4.7 设置右上角控件

除了“最大化”和“最小化”按钮外，还可以将一个控件设置在右上角。通常情况下，放置在右上角的控件可以是“关闭”按钮或者是一个菜单。例如，为了简单起见，这里只添加一个按钮，代码如下：

```
Button button = new Button(tabFolder, SWT.ARROW|SWT.RIGHT);
//设置显示在右上方的控件
tabFolder.setTopRight( button );
```

程序运行后，在右上角显示按钮的效果如图 6.25 所示。

使用 `setTopRight(Control control)` 方法设置的控件，默认为右对齐（`SWT.RIGHT`），若要使整个按钮充满，需使用 `setTopRight(Control control, int alignment)` 方法。例如，设置按钮充满整个右上角区域的代码如下所示：

```
tabFolder.setTopRight( button ,SWT.FILL);
```

充满整个右侧的选项卡效果如图 6.26 所示。如果此时想取消右上角的控件，只需要设置控件为“null”即可，例如：

```
tabFolder.setTopRight(null)
```

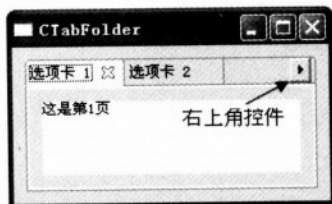


图 6.25 右上角带有按钮的选项卡

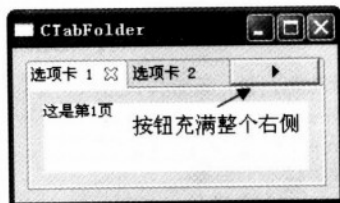


图 6.26 右侧充满状态的选项卡

可以在 Eclipse 的视图选项卡中找到设置了右上角控件为工具栏（ToolBar）的例子。如图 6.27 所示，为控制台视图右上方所设置的工具栏控件。

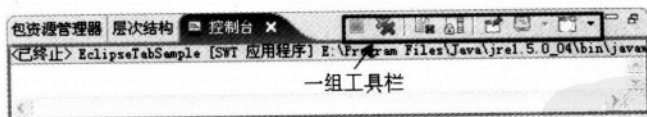


图 6.27 控制台视图

6.4.8 自定义选项的常用方法

自定义选项卡除了具有 `TabFolder` 的一些方法外，还扩展了一些常用的方法，如下：

- ❑ 设置是否显示了边框：`setBorderVisible(boolean show)`。`true` 为显示边框，`false` 为不显示边框。
- ❑ 设置选项卡字体：`setFont(Font font)`。

- ❑ 设置选项卡高度: `setTabHeight(int height)`。
- ❑ 设置选项卡的位置: `setTabPosition(int position)`, 默认为显示在上方, 若显示在下方, 则使用常量 `SWT.BOTTOM`; 若显示在上方, 则使用常量 `SWT.TOP`。

6.5 分割窗框 (SashForm)

分割窗框是将屏幕的区域分成几部分, 并且能够拖动窗框来改变窗口的大小。如图 6.28 所示为一个带有两个窗口的 (SashForm), 图 6.29 显示的是拖动状态下窗框的效果图。

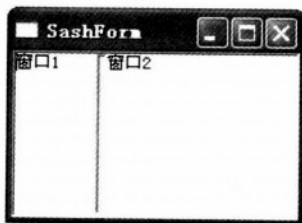


图 6.28 分割窗框

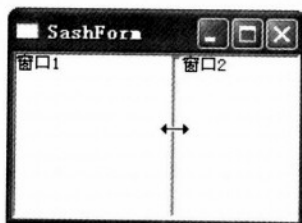


图 6.29 拖动状态中的窗框效果

创建如图 6.28 所示的窗框代码如下:

SashFormSample.java

```
//创建窗框对象, 设置样式为水平排列
SashForm form = new SashForm(shell, SWT.HORIZONTAL|SWT.BORDER);
form.setLayout(new FillLayout());
//创建窗口 1 的面板
Composite child1 = new Composite(form, SWT.NONE);
child1.setLayout(new FillLayout());
new Text(child1, SWT.MULTI).setText("窗口 1");
//创建窗口 2 的面板
Composite child2 = new Composite(form, SWT.NONE);
child2.setLayout(new FillLayout());
new Text(child2, SWT.MULTI).setText("窗口 2");
//设置初始状态两个面板所占的比例
form.setWeights(new int[] {30,70});
```

6.5.1 分割窗框的样式

除了带有边框显示的分割窗外, 还可以设置平滑外观的窗框, 使用 `SWT.SMOOTH` 常量。例如, 将上边的代码修改成如下, 显示的效果如图 6.30 所示。

```
SashForm form = new SashForm(shell, SWT.HORIZONTAL|SWT.SMOOTH);
```

也可以将窗口放置的位置改为垂直放置, 使用 `SWT.VERTICAL` 样式常量。例如, 将创建 SashForm 的代码改为:

```
SashForm form = new SashForm(shell, SWT.VERTICAL | SWT.BORDER);
```

程序运行后效果如图 6.31 所示。

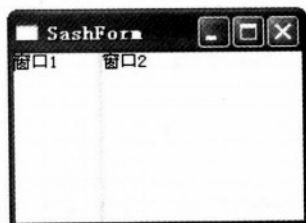


图 6.30 SWT.SMOOTH 样式效果图

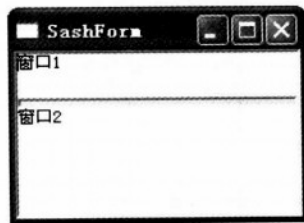


图 6.31 SWT.VERTICAL 样式效果图

6.5.2 设置窗框显示的比例

当显示多个窗口时，可以设置初始化状态时每个窗口的比例。例如，有 3 个窗口，从左到右所占的空间比例分别为 30%、30%和 40%。那么使用的代码如下：

```
form.setWeights(new int[] {30,30,40});
```

如图 6.32 所示为设置比例后的窗口示意图。

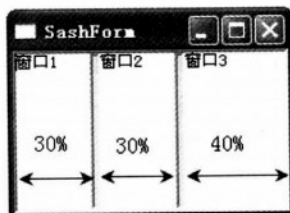


图 6.32 设置比例的窗口示意图

注意：设置比例时，int[]数组的长度要等于窗框所包含控件的个数，否则会出现运行错误。

6.5.3 设置窗框最大化所显示的控件

分割窗框中提供了一个可以设置最大化时显示某一个控件的方法——setMaximizedControl(Control control)。下面以 6.4.5 节中的例子，将选项卡中“最大化”按钮同 SashForm 中的设置最大化控件的方法结合起来。

该程序的大致原理是，创建一个 SashForm，左边的窗框放置一个面板，面板上有一个标签，右侧的窗框上放置选项卡，当单击选项卡的“放大”按钮后，使用以下代码设置窗框最大化时显示的控件为选项卡：

```
form.setMaximizedControl(folder);
```

当单击“还原”按钮时，又将最大化显示的控件置为 null，回到初始状态，代码如下：

```
form.setMaximizedControl(null);
```

图 6.33 和图 6.34 显示了设置最大化显示时不同的界面效果。

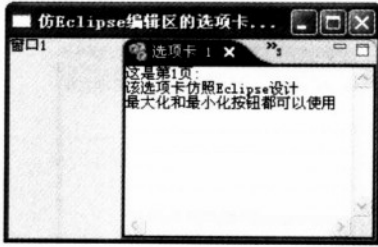


图 6.33 setMaximizedControl(null)时的效果



图 6.34 setMaximizedControl(folder)时的效果

由于该程序与 6.4.5 中的代码实现差不多，只是修改了一些事件处理，所以在这里就不多解释了，该程序的详细代码请参见本书所附光盘部分中的 EclipseTabSample2.java 文件。

通过此程序只想告诉读者，可以结合其他的控件来利用 SashForm。分割窗框这种可以放大充满整个窗口的效果，也是在布局设计时常用的容器之一。

6.6 自定义分割框（CBanner）

自定义分割框更像是 SashForm 的变体，它有 3 个控件，分别放在窗口的左侧（left）、右侧（right）和底部（bottom）。左侧和右侧的窗口可以改变大小，但底部的大小是不可以变化的。如图 6.35 所示为一个简单的 CBanner 容器。

创建该程序的代码如下：

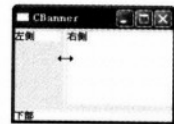


图 6.35 CBanner 效果示意图

CBannerSample.java

```
//创建 CBanner 对象
CBanner banner = new CBanner(shell,SWT.BORDER);
banner.setLayout(new FillLayout());

//创建 3 个面板，分别放置到左侧，右侧和下部
Composite left = new Composite(banner,SWT.NONE);
left.setLayout(new FillLayout());
new Text(left,SWT.MULTI).setText("左侧");

Composite right = new Composite(banner,SWT.NONE);
right.setLayout(new FillLayout());
new Text(right,SWT.MULTI).setText("右侧");

Composite bottom = new Composite(banner,SWT.NONE);
bottom.setLayout(new FillLayout());
new Text(bottom,SWT.MULTI).setText("下部");

//设置左侧的控件
```



```
banner.setLeft(left);  
//设置右侧的控件  
banner.setRight( right);  
//设置下部的控件  
banner.setBottom( bottom );
```

6.6.1 改变分割线的外观

另外通过设置 `setSimple(boolean simple)` 方法，可以改变左右窗框分割线的外观，例如，在以上程序的代码中添加如下代码：

```
banner.setSimple(false);
```

程序再次运行后，效果如图 6.36 所示。

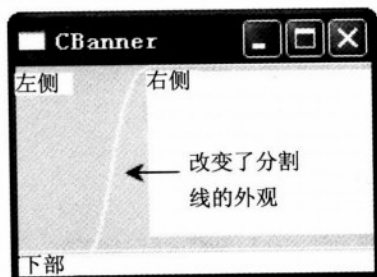


图 6.36 `setSimple(false)` 状态下的样式

6.6.2 Eclipse 中的 CBanner

其实，不难在 Eclipse 的工作区中找到 CBanner 的身影。如图 6.37 所示，在 Eclipse 工具栏与透视图的布局中，选择使用的就是 CBanner。

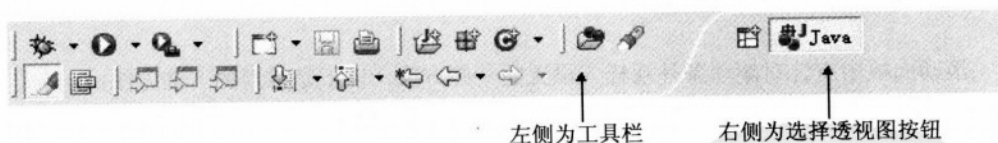


图 6.37 Eclipse 中使用的 CBanner

6.7 滚动面板 (ScrolledComposite)

滚动面板，顾名思义是带有垂直滚动条和水平滚动条的面板。如图 6.38 所示，即为一个普通的滚动面板。



图 6.38 滚动面板示意图

创建如图 6.38 所示的滚动面板的代码如下：

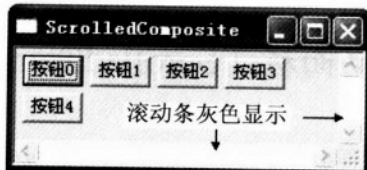
```
//创建一个滚动面板对象
final ScrolledComposite sc = new ScrolledComposite(shell, SWT.H_SCROLL | SWT.V_SCROLL |
SWT.BORDER);
//创建一个普通的面板
final Composite c = new Composite(sc, SWT.NONE);
GridLayout layout = new GridLayout();
layout.numColumns = 4;
c.setLayout(layout);
for (int i=0;i<20;i++)
{
    Button bt =new Button (c, SWT.PUSH);
    bt.setText("按钮"+i);
    c.setSize(c.computeSize(SWT.DEFAULT, SWT.DEFAULT));
}
//将普通面板设置为受控的滚动面板
sc.setContent(c);
```

在使用滚动面板的过程中,设置某一个控件受控于滚动面板的方法是 `setContent(Control content)`。

6.7.1 设置滚动条的样式

滚动面板中有针对滚动条外观样式设置的一些方法,以下是常用的一些方法:

- ❑ 设置是否总是显示滚动条: `setAlwaysShowScrollBars(boolean show)`。如果设置为 `true`,则即使窗口变小,不需要滚动条时,滚动条也会以灰色来显示,如图 6.39 所示。如果设置为 `false`,则当不需要滚动条时,就不会显示滚动条。

图 6.39 `setAlwaysShowScrollBars(true)`时的效果

- ❑ 设置水平滚动条是否显示: `setExpandHorizontal(boolean expand)`, 默认为 `false`。
- ❑ 设置绘制滚动条是否显示: `setExpandVertical(boolean expand)`, 默认为 `false`。如果设置为 `true`, 则即使窗口缩小, 也不出现水平或垂直的滚动条。如图 6.40 和图 6.41 所示为分别取消水平滚动条和垂直滚动条的效果图。

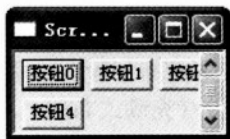


图 6.40 不显示水平滚动条



图 6.41 不显示垂直滚动条

6.7.2 滚动面板的其他方法

除了设置滚动条的一些方法外, 滚动面板类 (`ScrolledComposite`) 还有如下一些常用的方法。

- ❑ 设置最小显示滚动条的高度: `setMinHeight(int height)`。默认为 0, 该方法只有当 `setExpandVertical(true)` 时才起作用。
- ❑ 设置最小显示滚动条的宽度: `setMinWidth(int width)`。默认为 0, 该方法只有当 `setExpandHorizontal(true)` 时才起作用。
- ❑ 设置最小显示滚动条的高度和宽度: `setMinSize(Point size)` 或 `setMinSize(int width, int height)`。这两个方法也只有当设置了 `setExpandVertical(true)` 和 `setExpandHorizontal(true)` 后才起作用。

6.8 本章小结

本章主要学习了常见的一些容器类, 其中面板类 (`Composite`) 是所有容器的父类, 也是最常用的容器。分组框和滚动面板也是经常用到的容器, 分割窗框和 `CBanner` 容器在整个框架的设计时有很大用处, 选项卡容器稍微复杂, 尤其是要理解 `TabFolder` 与 `TabItem` 之间的关系, 对以后学习表格和树等控件打下了基础。最后自定义选项卡 (`CTabFolder`) 功能强大并且可设置多样的外观, 在 `Eclipse` 工作台中也可以发现它的身影。

第 7 章 SWT 布局管理器

本章将首先从总体上介绍 SWT 布局管理器的知识，了解布局管理的意义，然后进一步理解 5 种常见的布局以及它们之间的不同之处：FillLayout（充满式布局）、RowLayout（行列式布局）、GridLayout（网格式布局）、FormLayout（表格式布局）和 StackLayout（堆栈式布局），接下来仿照 Swing 中的 BorderLayout 布局讲述如何创建自定义布局类。最后，使用 Visual Editor 可视化的环境进行不同布局的设置。

7.1 布局管理器概述

在上文中讲到过面板类（Composite）也可以理解为容器类。所谓容器类，就是可以包含一些基本的控件，如按钮、标签等。常见的面板类有：Shell 类、Group 类、TabFolder 类等。那么放入到面板类中的控件是如何设置大小和位置的呢？SWT 提供了两种定位控件位置和大小方法：绝对定位和托管定位。

7.1.1 绝对定位

为每个控件设置明确的 X 和 Y 位置，并通过代码设置一定的宽度和高度。设置控件的位置和大小的方法是 `setBounds(int x,int y,int width,int height)`。例如有这样一个按钮，代码如下：

```
Button button= new Button( shell , SWT.NONE );
button.setBounds( 30,20, 100,50);
button.setText("(30, 20, 100,50)");
```

该按钮在窗口中显示的位置如图 7.1 所示。

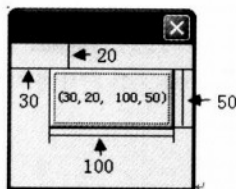


图 7.1 绝对定位的按钮

7.1.2 托管定位

不必为每个控件设置大小，所做的工作只是将控件放入到面板中，至于控件以何种方

式在面板排列,要依据所设定的布局(Layout)来管理。常见的布局有 FillLayout、RowLayout、GridLayout、FormLayout 和 StackLayout 5 种。

设定面板布局的方法是 setLayout(Layout layout)。例如,为 Shell 对象创建 FillLayout 布局的代码如下所示:

```
Shell shell = new Shell();
shell.setLayout(new FillLayout());
```



此外,即使对面板设置了布局,还可以为每个控件设置布局数据类(layoutData),习惯上,布局数据类通过取代布局类类名中的 Layout 为 Data 的标识。例如,布局类 RowLayout 具有布局数据类 RowData。布局类 GridLayout 使用布局数据类 GridData,而布局类 FormLayout 具有名为 FormData 的布局数据类。

例如,为一个按钮设置布局数据类的代码如下,该布局类保证了该按钮的长为 50,宽为 10。

```
Button button = new Button(shell, SWT.PUSH);
button.setLayoutData(new RowData(50, 10));
```

表 7.1 所示为绝对定位和托管定位控件的比较。

表 7.1 绝对定位与托管定位的比较

	初 始 状 态	调整窗口大小后	说 明
绝对定位			使用绝对定位,控件的位置和大小都是不变的,即使窗口大小调整后,也不会影响布局
托管定位 (使用 FillLayout 布局)			使用托管定位,控件的大小和位置都是通过布局管理器设定的,当窗口大小改变时,会重新计算控件的大小和位置

为了更好地理解布局,可以打一个形象的比喻来说明这两种定位布局的方式。把容器比喻成一个黑箱子,而各种控件比喻成不同的小球。如果使用绝对定位,把小球放到黑箱子之前就已经知道小球所摆放的位置了,那么小球放入到黑箱子后,位置是固定的。如果使用托管定位,可以理解这个黑箱子并不是空的,可能有不同的布局可以将小球分开,这时只需要将小球依次放入到箱子中,不需要知道具体放到哪个位置,而最终小球的位置要看箱子中的布局是什么样的。布局管理器可以理解为箱子的构造,对于不同构造的箱子,放入到其中的小球位置是不一样的。

7.1.3 常见的布局管理器

布局管理器是 SWT 界面设计中很重要的一项基础知识,读者一定要区分不同布局之间的差别。所有的布局都是 `org.eclipse.swt.widgets.Layout` 的子类, `Layout` 类是一个抽象类,每个具体的布局都要继承 `Layout` 类。SWT 中常用的布局类有以下几种。

- ❑ `FillLayout` (充满式布局): 在单行或者单列中放置相同大小的控件,是最简单的布局。
- ❑ `RowLayout` (行列式布局): 在单行或者多行中放置控件,应用了 `fill`、`wrap` 和 `spacing` 等选项。
- ❑ `GridLayout` (网格格式布局): 像表格一样放置控件,功能很强大,几乎能满足各种布局需要。
- ❑ `FormLayout` (表格式布局): 与 `GridLayout` 功能差不多的布局,可以通过定义 4 个边的“附加值”来放置控件。
- ❑ `StackLayout` (堆栈式布局): 类似堆栈式的布局,只显示最上方的控件。

⚠注意: `FillLayout`、`RowLayout`、`GridLayout`、`FormLayout` 在 `class org.eclipse.swt.layout.*` 包中, `StackLayout` 在 `org.eclipse.swt.custom.*` 包中。

图 7.2 是布局类的继承关系图。

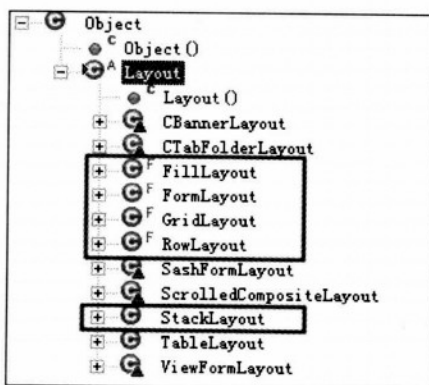


图 7.2 布局类的类继承关系图

从图中还可以看到其他的布局类,如 `TableLayout`、`ViewFormLayout` 等布局,这些布局主要是针对特定的容器类布局的,在讲述不同布局时还会用到,这里不多作介绍了。

7.2 FillLayout (充满式布局)

`FillLayout` 是充满式布局,也是最简单的布局。它设置控件布局的规则是:试图填充一

行或一列，尽可能地充满整个面板，并且强制所有控件平均分配大小。FillLayout 不会自动折行，也不能设置每个控件之间的空隙，但能够指定面板的四周的补白。创建一个 FillLayout 的代码如下：

```
FillLayout layout = new FillLayout( SWT.VERTICAL );
```

或

```
FillLayout layout = new FillLayout();
layout.type=SWT.VERTICAL;
```

📌注意：若不指定 type，默认为 SWT.HORIZONTAL 水平填充。

7.2.1 水平填充（默认）和垂直填充

下面以 3 个按钮为例，分别以水平方式和垂直方式填充到 Shell 面板中，水平填充时的代码如下，若改成垂直填充，只需要改变 layout.type=SWT.VERTICAL。

FillLayoutSample.java

```
Shell shell = new Shell(display,SWT.SHELL_TRIM);




FillLayout layout = new FillLayout();
layout.type=SWT.HORIZONTAL;
shell.setLayout( layout );

new Button( shell , SWT.NONE ).setText("B1");
new Button( shell , SWT.NONE ).setText("Button2");
new Button( shell , SWT.NONE ).setText("B3");

shell.layout();
shell.pack();
shell.open();
```

表 7.2 所示为两种填充方式的不同布局效果。

表7.2 两种填充方式的比较

	初 始 状 态	调整窗口大小后	说 明
水平填充 layout.type=SWT.HORIZONTAL（默认）			使用水平填充，控件会依次从左到右填充，并且试图填满整个面板
垂直填充 layout.type= SWT.VERTICAL			使用垂直填充，空间会依次从上到下填充

7.2.2 设置四周补白

如果在布局中设置四周补白，则代码如下：

```
FillLayout layout = new FillLayout();
layout.type=SWT.VERTICAL;
layout.marginHeight=10;//设置上下补白高度
layout.marginWidth=20;//设置左右
layout.spacing=5;//设置控件之间的空隙
shell.setLayout( layout );
```

图 7.3 所示为设置了补白后的布局。即使调整窗口大小，四周的补白和控件之间的间隙大小也不会改变。图 7.4 所示为调整窗口大小后的布局。

窗口大小调整后，marginWidth、marginHeight 和 spacing 的大小都未改变，如图 7.4 所示。

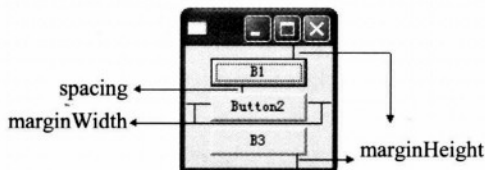


图 7.3 设置补白和间隙后的布局

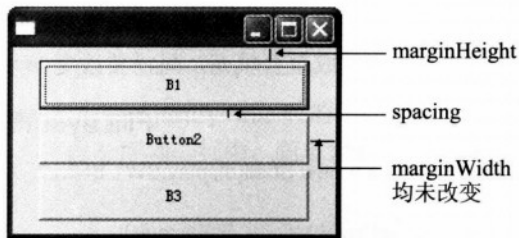


图 7.4 调整窗口后补白和间隙不变

7.3 RowLayout (行列式布局)

RowLayout 比 FillLayout 复杂一些，RowLayout 填充控件时可以折行显示，并且可以使用 RowData 设置某一个指定控件的大小。创建一个 RowLayout 布局的代码如下：

RowLayoutSample.java

```
Shell shell = new Shell(display,SWT.SHELL_TRIM);

RowLayout layout = new RowLayout();
layout.type = SWT.HORIZONTAL;//设置水平填充
layout.marginLeft = 5;//左补白
layout.marginTop = 5;//上补白
layout.marginRight = 5;//右补白
layout.marginBottom = 5;//下补白
layout.spacing = 2;//控件的间隙
shell.setLayout( layout );

new Button( shell , SWT.NONE ).setText("B1");
new Button( shell , SWT.NONE ).setText("Button2");
new Button( shell , SWT.NONE ).setText("Wide Button3");
```



```
new Button( shell , SWT.NONE ).setText("B4");  
shell.layout();
```

该代码运行后的布局效果如图 7.5 所示。

从图 7.5 中可以看出,与 FillLayout 不同,默认情况下,每个控件的大小是不一样的,不是平均分配的,而是依照控件所合适的大小来设定的。

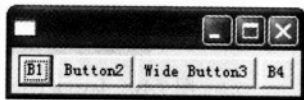


图 7.5 RowLayout 布局示例

7.3.1 设置折行显示: wrap 属性

wrap 属性可设置折行显示,默认值为 true。图 7.6 和图 7.7 比较了 wrap=true 和 wrap=false 时的不同效果。

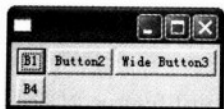


图 7.6 layout.wrap=true

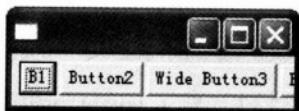


图 7.7 layout.wrap=false

从以上两个图中不难看出,设置折行显示时,一行的大小不够时就自动转向下一行。而设置为 false 时,即使一行不够,也不会将控件转到下一行显示。

7.3.2 设置空间大小: pack 属性

pack 属性自动设置空间大小,默认为 true。

```
layout.pack=false;//默认为 true
```

图 7.8 和图 7.9 比较了 pack=true 和 pack=false 时的不同效果。从图中可以看出,设置 layout.pack=false 后,控件会平均分配大小,有点类似 FillLayout。

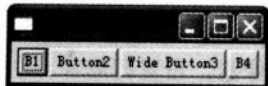


图 7.8 layout.pack=true

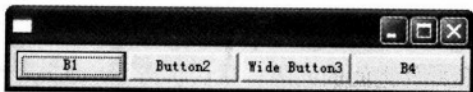


图 7.9 layout.pack=false

7.3.3 设置填充方式: type 属性

type 属性用来设置填充方式是水平填充还是垂直填充,默认为水平填充。

```
layout.type = SWT.VERTICAL;//设置垂直方式,默认为水平 SWT.HORIZONTAL
```

图 7.10 和图 7.11 比较了 layout.type = SWT.VERTICAL 和 layout.type = SWT.HORIZONTAL 时的不同效果。

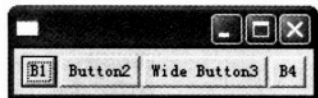


图 7.10 layout.type = SWT. HORIZONTAL

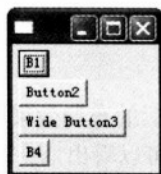


图 7.11 layout.type = SWT. VERTICAL

从以上两个图中可以看出，设置 `layout.type = SWT. VERTICAL` 后，控件按照垂直的方式填充。

7.3.4 设置是否充满整行：justify 属性

`justify` 属性用来判断是否充满整个一行，默认为 `false`。

```
layout.justify=true;//默认为 false
```

图 7.12 和图 7.13 比较了 `layout.justify=false` 和 `layout.justify=true` 时的不同效果。从图中可以看出，设置 `layout.justify=true` 后，控件是平均分配到一行中，随着窗口大小的改变，间距也在变。

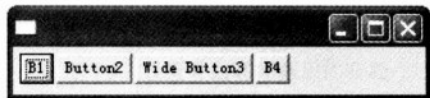


图 7.12 layout.justify=false

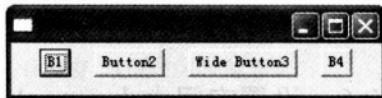


图 7.13 layout.justify=true

⚠注意：如果设置了 `layout.justify=true` 后，再设置 `layout.spacing` 属性，如果这时窗口足够大，该属性将不起作用。

7.3.5 设置补白和间隔

`marginLeft`、`marginTop`、`marginRight`、`marginBottom` 和 `spacing` 属性的作用是设定四周的补白和控件之间的间隔。默认状态下，这几个值大小为 3 像素。设置四周的补白和间隔都为 10 像素的代码如下：

```
layout.marginBottom=10;
layout.marginTop=10;
layout.marginLeft=10;
layout.marginRight=10;
layout.spacing=10;
```

图 7.14 所示为这几个属性的示意图。

另外，与 `FillLayout` 一样，`RowLayout` 也有补白高度 (`marginHeight`) 和补白宽度 (`marginWidth`) 属性，只不过在 `RowLayout` 布局中，`marginBottom` 和 `marginTop` 组合为 `marginHeight`，`marginLeft` 和 `marginRight` 组合为 `marginWidth`。具体区别请比较图 7.3 和

图 7.14。

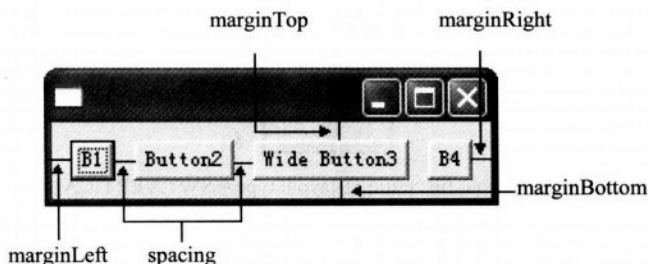


图 7.14 补白和间隔示意图

7.3.6 设置控件的大小 RowData

使用 RowData 对象，设置某一个控件的大小。为一个按钮设置大小的代码如下：

```
Button b = new Button( shell , SWT.NONE );
b.setText("B1");
//设置按钮宽为 100 像素，高为 30 像素
b.setLayoutData( new RowData(100,30));
```

这样，设定大小后的按钮程序运行后效果如图 7.15 所示。

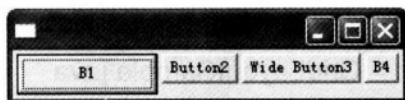


图 7.15 设定 RowData 大小后的按钮

7.3.7 设置是否等宽或等高：fill 属性

当以水平方式填充时，fill 属性试图使所用控件具有同样高度；当以垂直方式显示时，试图使用所有控件具有同样宽度。默认为 false。

```
layout.fill = true; //默认为 false
```

在上面的程序中，使用了 RowData 对按钮“B1”进行了大小的设置，而其他按钮并没有设置具体的大小，这时，设置 layout.fill = true，效果如图 7.16 所示。

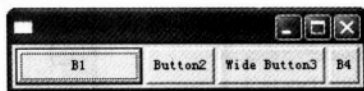


图 7.16 layout.fill = true

从图 7.15 和图 7.16 的比较中可以看出，如果设置了 layout.fill = true，则布局管理器试图使所有的按钮同高。若此时设置 layout.type = SWT.VERTICAL，垂直填充时，布局管理

器则试图使所有的按钮同宽，效果如图 7.17 所示。



图 7.17 layout.fill = true 且 layout.type = SWT.VERTICAL

7.4 GridLayout (网格式布局)

GridLayout 网格式布局是功能很强大的布局，类似于网页中使用的表格。使用 GridLayout 布局，控件将会按照网格的方式进行填充。与 RowLayout 一样，GridLayout 所放置的控件可以有一个关联的布局数据对象 GridData。GridLayout 的强大功能在于，可以使用 GridData 为每一个控件设置不同的布局。

7.4.1 设置网格的列数：numColumns 属性

numColumns 属性设置了网格的列数，默认为 1 列。控件从左到右放置在不同的列中，当一列填满后，会自动填充下一列。以下代码显示的是创建一个 3 列的网格布局。

GridLayoutSample.java

```
GridLayout gridLayout = new GridLayout();  
gridLayout.numColumns = 3;  
shell.setLayout(gridLayout);  
new Button(shell, SWT.PUSH).setText("B1");  
new Button(shell, SWT.PUSH).setText("Wide Button 2");  
new Button(shell, SWT.PUSH).setText("Button 3");  
new Button(shell, SWT.PUSH).setText("B4");  
new Button(shell, SWT.PUSH).setText("Button 5");  
new Button(shell, SWT.PUSH).setText("B6");
```

代码运行后显示的布局效果如图 7.18 所示。

若将 numColumns 属性设置为 2 列，则布局改变为如图 7.19 所示。

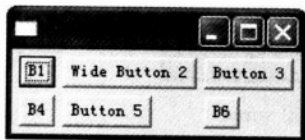


图 7.18 numColumns=3

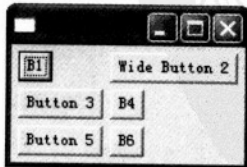


图 7.19 numColumns=2

7.4.2 设置网格等宽: makeColumnsEqualWidth 属性

设置 `makeColumnsEqualWidth=true` 可以强制让所有列都具有相同的宽度，默认值为 `false`。如果在 `GridLayoutSample.java` 代码中作以下设置：

```
gridLayout.makeColumnsEqualWidth=true;
```

代码运行后显示的布局如图 7.20 所示。

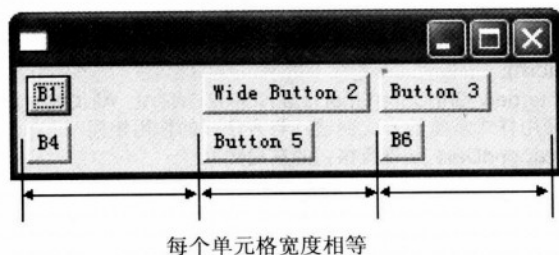


图 7.20 `makeColumnsEqualWidth=true`

7.4.3 设置补白和间隔

与 `RowLayout` 类似, `GridLayout` 也有 `marginLeft`、`marginTop`、`marginRight` 和 `marginBottom` 等属性。与之不同的是 `GridLayout` 将间隔 (`spacing`) 分为水平间隔 (`horizontalSpacing`) 和垂直间隔 (`verticalSpacing`)，默认为 5 像素大小。设置水平间隔和垂直间隔的代码如下：

```
gridLayout.verticalSpacing=10;  
gridLayout.horizontalSpacing=10;
```

水平间隔和垂直间隔的示意图如图 7.21 所示。

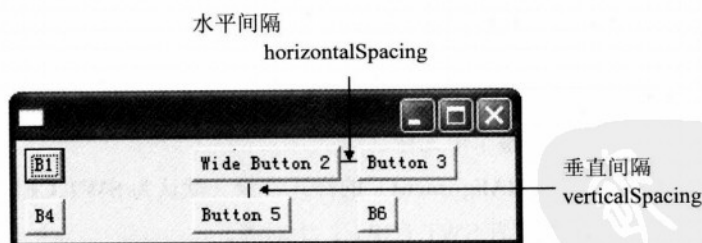


图 7.21 水平间隔与垂直间隔示意图

7.4.4 使用 GridData 对象

`GridData` 与 `RowData` 对象类似, 是设置 `GridLayout` 布局中的某一个控件, 但比 `RowData`

功能强大。例如，为一个按钮设置 GridData 的代码如下：

1. 方法一

```
Button button = new Button(shell, SWT.PUSH);
button.setText("Button");
GridData gridData = new GridData();//创建 GridData 对象
gridData.horizontalAlignment=GridData.FILL;//设置水平填充方式为充满单元格
gridData.grabExcessHorizontalSpace = true;//设置水平抢占空间的方式
button.setLayoutData( gridData );//设置按钮的布局数据
```

2. 方法二

```
Button button = new Button(shell, SWT.PUSH);
button.setText("Button");
button.setLayoutData(new GridData(GridData.HORIZONTAL_ALIGN_FILL |GridData.GRAB_
HORIZONTAL));//使用样式常量的方式创建，与方法一的作用相同
button.setLayoutData( gridData );//设置按钮的布局数据
```

虽然有两种方法都可以创建 GridData 对象，但从代码的整齐角度来看，建议使用第 2 种方法。

🔔注意：不要重用 GridData 对象。每一个面板（Composite）对象中被 GridLayout 管理的控件必须有一个唯一的 GridData 对象。如果在设置布局时一个 GridLayout 中的控件的 GridData 为 null，就会为它创建一个唯一的 GridData 对象。

7.4.5 设置单元格对齐方式：horizontalAlignment 和 verticalAlignment 属性

horizontalAlignment 和 verticalAlignment 分别为单元格设置水平和垂直方向的对齐方式。可选的样式常量如下：

1. 水平对齐方式（horizontalAlignment）的样式常量（默认为 SWT.BEGINNING）

- ☐ SWT.BEGINNING（或者 SWT.LEFT）：左对齐。
- ☐ SWT.CENTER：水平居中。
- ☐ SWT.END（或者 SWT.RIGHT）：右对齐。
- ☐ SWT.FILL：水平充满整个单元格。

2. 垂直对齐方式（verticalAlignment）的样式常量（默认为 SWT.CENTER）

- ☐ SWT.BEGINNING（或者 SWT.TOP）：上对齐。
- ☐ SWT.CENTER：垂直居中。
- ☐ SWT.END（或者 SWT.BOTTOM）：下对齐。
- ☐ SWT.FILL：垂直充满整个单元格。

下面对 GridLayoutSample.java 程序中的第 5 个按钮进行改造。代码如下：

```
//new Button(shell, SWT.PUSH).setText("Button 5");
//改为以下代码
```

```

Button bt5 = new Button(shell, SWT.PUSH);
bt5.setText("Button 5");
GridData gridData = new GridData();
//设置水平对齐方式
gridData.horizontalAlignment=SWT.BEGINNING;
//为第 5 个按钮设置 GridData 对象
bt5.setLayoutData( gridData );

```

图 7.22~图 7.25 显示了不同对齐方式下，第 5 个按钮（Button 5）的水平对齐的方式。

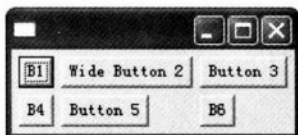


图 7.22 SWT.BEGINNING（默认）

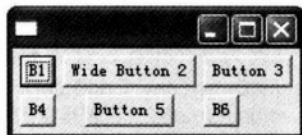


图 7.23 SWT.CENTER

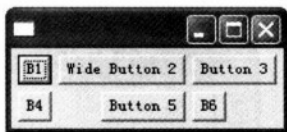


图 7.24 SWT.END

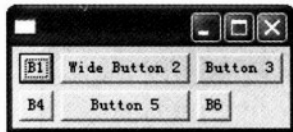


图 7.25 SWT.FILL

7.4.6 设置缩进大小：horizontalIndent 和 verticalIndent 属性

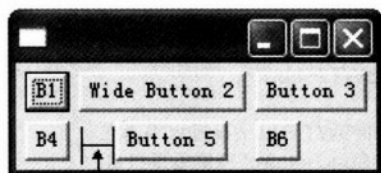
horizontalIndent 和 verticalIndent 属性可以设置控件水平和垂直方向缩进的像素大小。默认值为 0。如果将第 5 个按钮作以下设置：

```

GridData gridData = new GridData();
gridData.horizontalIndent=20;//设置缩进为 20 像素大小
bt5.setLayoutData( gridData );

```

代码运行后的效果如图 7.26 所示。



缩进值

图 7.26 水平缩进示意图

7.4.7 设置单元格跨行和跨列显示：horizontalSpan 和 verticalSpan 属性

horizontalSpan 和 verticalSpan 属性可以设置单元格水平跨越和垂直跨越的单元格数，是

控制布局属性中很重要的一个属性。例如，将第 5 个按钮的 `horizontalSpan` 设置为 2，并且为充满状态，代码如下：

```
GridData gridData = new GridData();
gridData.horizontalSpan=2;//设置水平跨越两个单元格
gridData.horizontalAlignment=SWT.FILL;
bt5.setLayoutData( gridData );
```

代码运行后的效果如图 7.27 所示。

从图中可以看出，Button 5 水平占据了两个单元格的位置。若将第 3 个按钮的 `verticalSpan` 设置为 2，并且设置为充满状态，代码如下：

```
GridData gridData = new GridData();
gridData.verticalSpan=2;//设置垂直跨越两个单元格
gridData.verticalAlignment=SWT.FILL;
bt3.setLayoutData( gridData );
```

代码运行后如图 7.28 所示，从图中可以看出，Button 3 垂直方向占据了两个单元格。

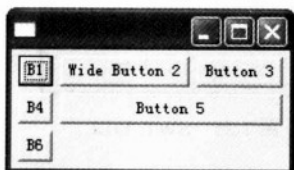


图 7.27 `horizontalSpan=2`

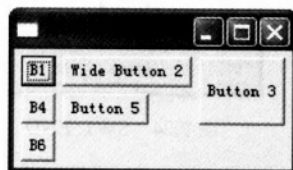


图 7.28 `verticalSpan=2`

7.4.8 设置单元格空间的抢占方式：`grabExcessHorizontalSpace` 和 `grabExcessVerticalSpace` 属性

`grabExcessHorizontalSpace` 和 `grabExcessVerticalSpace` 属性可以设置当面板变化时，单元格所抢占空间的方式，默认值为 `false`。什么是抢占空间呢？还是以例子作一下介绍。如上个例子中，对第 3 个按钮作以下设置：

```
GridData gridData = new GridData();
gridData.verticalSpan=2;
gridData.verticalAlignment=SWT.FILL;
//新加属性
gridData.horizontalAlignment=SWT.FILL;//设置水平填充
gridData.grabExcessVerticalSpace=true;//设置垂直抢占
gridData.grabExcessHorizontalSpace=true;//设置水平抢占
bt3.setLayoutData( gridData );
```

图 7.29 和图 7.30 分别所示了当窗口变大时，未设置抢占和设置了抢占空间属性的不同效果图。

从两个图中可以看出，如果没有设置抢占空间，当窗口变大时，Button 3 并没有任何大小变化；而设置了抢占空间，Button 3 会随着窗口的变大而增大，这种布局很重要，对设定随着窗口大小改变而改变的控件很有用。

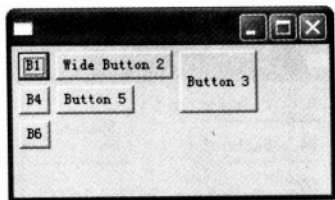


图 7.29 未设置抢占空间

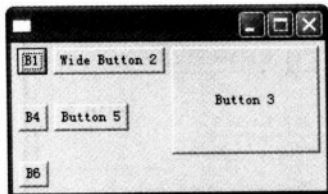


图 7.30 设置了抢占空间

通常在一个应用程序窗口中，需要有至少一个抢占式的控件。如果多于一个窗口试图抢占同样的空间，那么额外的空间会被均匀地分配到抢占式的控件中。

7.4.9 设置的控件大小：minimumWidth 和 minimumHeight 属性

minimumWidth 和 minimumHeight 属性可以设置控件最小的宽度和高度。注意，这两个属性仅当 grabExcessHorizontalSpace=true 和 grabExcessVerticalSpace=true 时才生效。例如，在以上的代码中作以下设置：

```
gridData.minimumHeight=100;//最小高度
gridData.minimumWidth=100;//最小宽度
```

代码运行后的效果如图 7.31 所示。

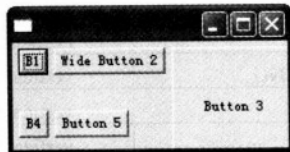


图 7.31 minimumHeight=100 且 gridData.minimumWidth=100

设置了 minimumWidth 和 minimumHeight 属性的 Button 3 即使窗口变小了，也不会超过这个最小值的大小。

7.4.10 设置控件大小：widthHint 和 heightHint 属性

widthHint 和 heightHint 属性可以设置控件的长和宽。与 minimumWidth 和 minimumHeight 所不同的是，即使设置 widthHint 和 heightHint 属性，当窗口变大时，控件的大小也会随着窗口的改变而改变。例如，上面例子中的代码改为：

```
gridData.widthHint=100;//设置宽度
gridData.heightHint=100;//设置高度
```

代码运行后，如图 7.32 所示。调整窗口大小后，效果如图 7.33 所示。

从以上两个图的比较中可以看出，设置了 widthHint 和 heightHint 属性后只是在程序刚一运行时才会起作用，而随着窗口改变，会重新计算控件的大小。

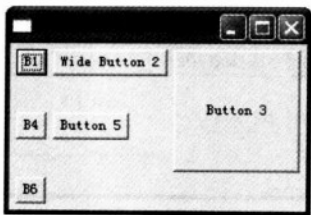


图 7.32 程序运行后的初始状态

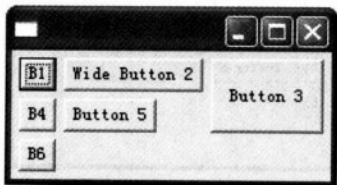


图 7.33 调整大小后的状态

7.4.11 样式常量对照表

前文提到过，创建 GridData 的第二种方式是使用样式常量，表 7.3 列举了样式常量所对应的属性值。

表7.3 样式常量对照表

样 式 常 量	对应属性值
GRAB_HORIZONTAL	grabExcessHorizontalSpace=true
GRAB_VERTICAL	grabExcessVerticalSpace=true
HORIZONTAL_ALIGN_BEGINNING	horizontalAlignment= SWT. BEGINNING
HORIZONTAL_ALIGN_CENTER	horizontalAlignment= SWT. CENTER
HORIZONTAL_ALIGN_END	horizontalAlignment= SWT.END
HORIZONTAL_ALIGN_FILL	horizontalAlignment= SWT.FILL
VERTICAL_ALIGN_BEGINNING	verticalAlignment=SWT. BEGINNING
VERTICAL_ALIGN_CENTER	verticalAlignment=SWT. CENTER
VERTICAL_ALIGN_END	verticalAlignment=SWT.END
VERTICAL_ALIGN_FILL	verticalAlignment=SWT.FILL
FILL_BOTH	horizontalAlignment=SWT.FILL verticalAlignment =SWT.FILL

7.5 FormLayout（表格式布局）

与 GridLayout 不同，FormLayout 通过设置 FormData 四边的附加值（FormAttachment 对象）来设置控件的布局。一个附加值让一个控件指定的一边附加到父面板容器类（Composite）的位置或者其他控件上。所以，这种布局可以指定某两个控件的相对位置，并且能随着窗口的改变而改变的功能也是很强大的。创建一个 FormLayout 布局的代码如下：

FormLayoutSample.java

```
//创建 FormLayout 对象
FormLayout formLayout = new FormLayout();
formLayout.marginHeight=5;//设置上下补白为 5 像素
```

```
formLayout.marginWidth=5;//设置左右补白为 5 像素  
shell.setLayout(formLayout);
```

```
Button bt1 = new Button(shell, SWT.PUSH);  
bt1.setText("B1");  
//创建 FormData 对象  
FormData formData = new FormData();  
//设定控件的上边框的位置  
formData.top = new FormAttachment (30,70,60);  
//设定控件的下边框的位置  
formData.bottom = new FormAttachment (100,-5);  
bt1.setLayoutData(formData);
```

7.5.1 设置补白和间隔

与 GridLayout 一样, FormLayout 也有 marginLeft、marginTop、marginRight、marginBottom、marginWidth、marginHeight 和 spacing 等属性。与之不同的是 FormLayout 将间隔 (spacing) 属性默认为 0 像素大小。用法请读者参见 7.4.3 节。

7.5.2 使用 FormData 对象

学习了前面的几种布局后, 现在读者对 FormData 对象应该不难理解。FormData 针对的是某一个控件进行的设置, 在创建 FormData 对象时, 通常要指定 4 个边的位置。创建一个 FormData 对象的代码通常如下:

```
formData.top = new FormAttachment (0,60);//设置上边框位置  
formData.bottom = new FormAttachment (100,-5);//设置下边框位置  
formData.left = new FormAttachment (20,0);//设置左边框位置  
formData.right = new FormAttachment (100,-3);//设置右边框位置  
bt1.setLayoutData(formData);
```

该代码运行后, 各属性所设置的值如图 7.34 所示。

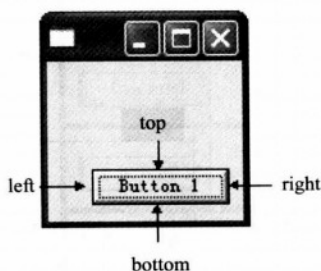


图 7.34 FormData 各属性示意图

设置 FormData 对象各属性时, 需要创建 FormAttachment 对象。FormAttachment 对象是具体某个边的位置设置, 所以理解 FormAttachment 对象对布局的设置很重要。

7.5.3 使用 FormAttachment 对象

以 7.5.2 节中的按钮为例，设置的左边框位置的代码如下：

```
formData.left = new FormAttachment (20,0);//设置左边框位置
```

这里创建的 FormAttachment 对象，使用的是 FormAttachment(int numerator, int offset)构造方法，numerator 是分子数（因为默认的分母是 100），所以也可以理解为所占的百分比，offset 是偏移量，如果是正数正向偏移，若是负数则反向偏移。图 7.35 显示了 numerator 参数所设置的位置示意图。

从图中可以看出，由于默认的分母值是 100，所以可以理解为是所占的百分比，当然也可以指定分子和分母的比例，那么就要使用另一个构造方法 FormAttachment(int numerator, int denominator, int offset)。例如，要使用比例为 30：70，则以上代码改为：

```
formData.left = new FormAttachment (30,70,0);
```

另外还有一个参数 offset 设置的是偏移的像素大小。例如，将以上代码中的 offset 值更改为 10 和 -10 时，效果如图 7.36 所示。

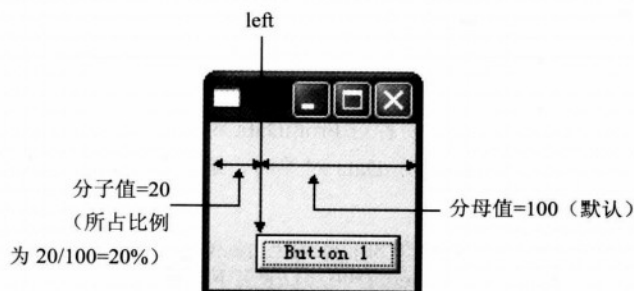


图 7.35 numerator 参数设置示意图

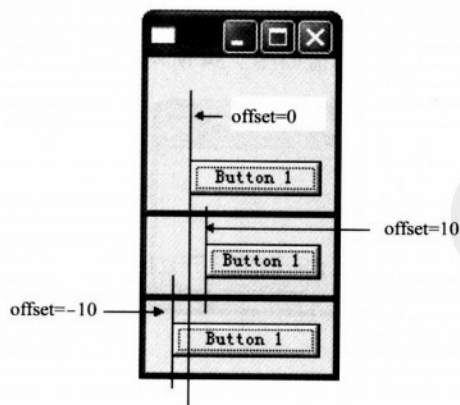


图 7.36 不同的 offset 值效果比较图

这里只对 `left` 属性的 `FormAttachment` 对象进行了详细的分析, 其他 `bottom`、`top` 和 `right` 属性也与之类似。

7.5.4 设置控件的相对位置

在使用 `FormAttachment` 对象时, 可以设置相对面板的位置, 也可以设置相对一控件的位置, 可以使用 `FormAttachment(Control control, int offset, int alignment)` 构造方法。Control 为所附加的控件对象, `offset` 是相对于控件偏移的像素数, `alignment` 使用样式常量, `SWT.TOP`、`SWT.BOTTOM`、`SWT.LEFT`、`SWT.RIGHT`、`SWT.CENTER` 和 `SWT.DEFAULT` 表示是按照附加控件的哪个边开始偏移位置的。

例如, 这里有两个按钮, 第一个按钮的位置依赖于面板, 而第二个按钮的位置依赖于第一个按钮。代码如下:

```
Button bt1 = new Button(shell, SWT.PUSH);
bt1.setText("Button 1");
FormData formData = new FormData();
formData.top = new FormAttachment (20,0);//设置上边框位置
formData.left = new FormAttachment (20,-10);//设置左边框位置
bt1.setLayoutData(formData);

Button bt2 = new Button(shell, SWT.PUSH);
formData = new FormData();
formData.top = new FormAttachment (20,0);//设置上边框位置
formData.left = new FormAttachment ( bt1, 20 ,SWT.RIGHT );//距 bt1 右边距 20 个像素
bt2.setLayoutData(formData);
bt2.setText("Wide Button 2");
```

该代码运行后, 效果如图 7.37 所示。调整窗口大小后, 如图 7.38 所示。



20 个像素

图 7.37 初始状态的布局

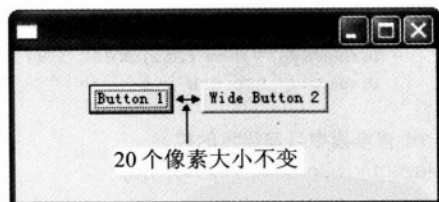


图 7.38 调整窗口后的布局

从以上两个图的比较中不难发现, 当控件设置 `FormAttachment` 对象时, 可以保证在窗口调整的情况下, 控件相对位置保持不变。这种特性也是 `FormLayout` 的强大之处。

7.6 StackLayout (堆栈式布局)

`StackLayout` 堆栈式布局类似于选项卡 (`TabFolder`), 当前只显示最上方的控件。例如,

面板中有 10 个文本框，面板设置为 StackLayout 布局，程序运行后如图 7.39 和图 7.40 所示，当单击“显示下一个文本框”按钮时，下一个文本框就显示出来。这样面板中始终只有一个文本框，设置最上方显示控件的属性是 layout.topControl。

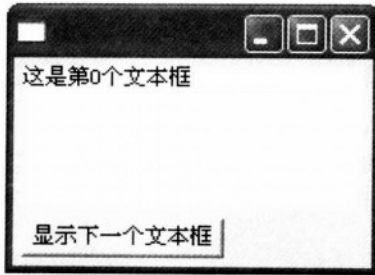


图 7.39 程序初始状态

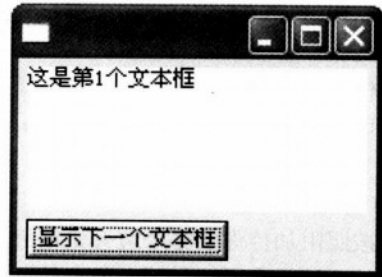


图 7.40 单击按钮后显示下一个文本框

该程序的实现代码如下：

StackLayoutSample.java

```
Shell shell = new Shell(display);
shell.setLayout(new GridLayout());
//创建放置文本框的面板
final Composite parent = new Composite(shell, SWT.NONE);
//设置面板的布局数据
parent.setLayoutData(new GridData(GridData.FILL_BOTH));
//创建堆栈式布局
final StackLayout layout = new StackLayout();
//将堆栈式布局应用于面板
parent.setLayout(layout);
//创建 10 个文本框
final Text[] textArray= new Text[10];
for (int i = 0; i < 10; i++) {
    textArray[i] = new Text(parent, SWT.MULTI);
    textArray[i].setText("这是第"+i+"个文本框");
}
//设置堆栈中当前显示的控件
layout.topControl = textArray[0];

Button b = new Button(shell, SWT.PUSH);
b.setText("显示下一个文本框");
//保存当前显示的文本框的索引值
final int[] index = new int[1];
//为按钮添加单击事件
b.addListener(SWT.Selection, new Listener(){
    public void handleEvent(Event e) {
        //计算出下一个文本框的索引数
        index[0] = (index[0] + 1) % 10;
        //设置当前显示的控件
        layout.topControl = textArray[index[0]];
    }
});
```



```
//重新刷新布局
parent.layout();
}
});

shell.setSize( 200,150);
```

堆栈式布局的用法虽然比较简单。但在界面元素很多的情况下，可以隐藏控件的这种方式，用处还是很大的。

7.7 自定义布局管理器

之前讲述的这些布局都是SWT中已经存在的布局，那么当按照自己的需要定制布局时，又该如何做呢？下面就来详细讨论一下自定义布局是如何实现的。

7.7.1 布局的基本原理

任何布局类都是 Layout 的子类，Layout 是一个抽象类，源代码如下：

```
package org.eclipse.swt.widgets;
import org.eclipse.swt.graphics.*;

public abstract class Layout {
    protected abstract Point computeSize (Composite composite, int wHint, int hHint, boolean
flush Cache);
    protected boolean flushCache (Control control) {
        return false;
    }
    protected abstract void layout (Composite composite, boolean flushCache);
}
```

创建一个自定义的布局类要继承 Layout 类，并且要实现 Layout 中的抽象方法。以下代码创建的是一个最简单的自定义类 MyLayout，这个类还没有作任何设置。

```
public class MyLayout extends Layout{
    //该方法计算面板显示的大小
    protected Point computeSize(Composite composite, int wHint, int hHint, boolean flushCache)
    {
        return new Point( wHint , hHint );
    }
    //设置子控件的位置
    protected void layout(Composite composite, boolean flushCache) {
    }
}
```

下面具体理解一下这两个方法的意义。

- ❑ `Point computeSize(Composite composite, int wHint, int hHint, boolean flushCache)`方法：该方法是计算布局的大小，也就是按照一定的规则算法计算出最终布局的长和宽，其中 `wHint` 和 `hHint` 是设置默认的宽和高。例如，当计算出来的长和宽小于默认的宽和高时，就可以使用默认的宽和高。`flushCache` 参数设置是否使用缓存的数据。
- ❑ `void layout(Composite composite, boolean flushCache)`方法：该方法是对该面板（参数 `composite`）中所有的控件（`Control`）设置显示的具体位置，通常获得该面板中的所有控件的方法是 `composite.getChildren()`。这样就可以根据指定的计算规则来放置每个控件的位置了。

综上所述，创建一个自定义布局关键是实现这两个方法，而具体布局的设置要根据设定的计算方法来实现。

7.7.2 布局计算的常用方法

涉及布局的计算方法，就需要总结一下有关面板（`Composite`）和控件（`Control`）两个类中有关位置的方法。

1. 控件（`Control`）类中的常用方法

- ❑ 计算控件合适的大小的方法：`Point computeSize(int wHint,int hHint)`和 `Point computeSize(int wHint, int hHint,boolean changed)`。例如：

```
Point point = control.computeSize(SWT.DEFAULT, SWT.DEFAULT);
int width=point.x;//宽度
int height=point.y//高度
```

- ❑ 获得控件当前坐标位置的方法：`Rectangle getBounds()`。例如：

```
Rectangle rect = control.getBounds();
int left = rect.x;
int right = rect.width;
int top = rect.y;
int bottom = rect.height;
```

- ❑ 设置控件的位置的方法：`setBounds(int x, int y, int width, int height)`或 `setBounds(Rectangle rect)`。

2. 面板（`Composite`）类中的常用方法

- ❑ 获得面板的大小区域的方法：`Rectangle getClientArea()`。
- ❑ 获得所有子控件的方法：`Control[] getChildren()`。
- ❑ 获得面板的布局对象：`Layout getLayout()`。

这些方法都是在计算控件布局时常用的方法，请读者熟练掌握。



7.7.3 自定义布局类 (BorderLayout)

下面以一个例子来具体说明自定义布局类是如何实现的。使用过 Swing 的读者一定对 BorderLayout 不陌生，它是这样一种布局，将控件按东、南、西、北、中 5 个区域放置，每个方向最多只能放置一个控件，随着窗口大小的改变，整个窗口会不断撑大。如图 7.41 和图 7.42 所示为窗口放大时，BorderLayout 布局的变化情况。

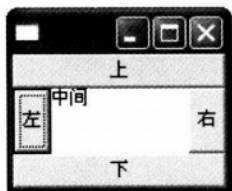


图 7.41 初始状态

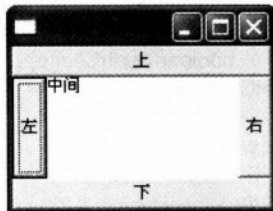


图 7.42 窗口放大后

从以上两图的比较中可以看出，随着窗口的变大，所有的控件也跟着变大，而且总是保持充满着整个窗口的状态。下面就来看一下这样的一个 BorderLayout 在 SWT 的布局中如何实现。

(1) 创建一个 BorderData 类，该类设置控件所在位置。该类实现的代码如下：

BorderData.java

```
package com.fengmanfei.ch7;

import org.eclipse.swt.SWT;

public final class BorderData {
    public int region = SWT.CENTER; // 默认为中间
    public BorderData() {
    }
    public BorderData(int region) {
        this.region = region;
    }
}
```

(2) 编写最重要的 BorderLayout 类，该类的详细代码如下所示，具体代码说明请参见代码的注释部分。

BorderLayout.java

```
package com.fengmanfei.ch7;

import org.eclipse.swt.SWT;
import org.eclipse.swt.graphics.Point;
import org.eclipse.swt.graphics.Rectangle;
import org.eclipse.swt.widgets.Composite;
```

```
import org.eclipse.swt.widgets.Control;
import org.eclipse.swt.widgets.Layout;

public class BorderLayout extends Layout {
    //定义存放在不同位置的 5 个控件
    private Control north;
    private Control south;
    private Control east;
    private Control west;
    private Control center;
    protected Point computeSize(Composite composite, int wHint, int hHint,
        boolean flushCache) {
        getControls(composite);
        // 定义面板的宽和高
        int width = 0, height = 0;
        // 计算面板的宽度
        width += west == null ? 0 : getSize(west, flushCache).x;
        width += east == null ? 0 : getSize(east, flushCache).x;
        width += center == null ? 0 : getSize(center, flushCache).x;
        // 如果上部和下部有控件, 则宽取较大的值
        if (north != null) {
            Point pt = getSize(north, flushCache);
            width = Math.max(width, pt.x);
        }
        if (south != null) {
            Point pt = getSize(south, flushCache);
            width = Math.max(width, pt.x);
        }

        // 计算面板的高度
        height += north == null ? 0 : getSize(north, flushCache).y;
        height += south == null ? 0 : getSize(south, flushCache).y;

        int heightOther = center == null ? 0 : getSize(center, flushCache).y;
        if (west != null) {
            Point pt = getSize(west, flushCache);
            heightOther = Math.max(heightOther, pt.y);
        }
        if (east != null) {
            Point pt = getSize(east, flushCache);
            heightOther = Math.max(heightOther, pt.y);
        }
        height += heightOther;

        // 计算的宽和高与默认的宽和高作比较, 返回之中较大的
        return new Point(Math.max(width, wHint), Math.max(height, hHint));
    }

    protected void layout(Composite composite, boolean flushCache) {
```

```

getControls(composite);
// 获得当前面板可显示的区域
Rectangle rect = composite.getClientArea();
int left = rect.x, right = rect.width, top = rect.y, bottom = rect.height;
// 将各个控件放置到面板中
if (north != null) {
    Point pt = getSize(north, flushCache);
    north.setBounds(left, top, rect.width, pt.y);
    top += pt.y;
}
if (south != null) {
    Point pt = getSize(south, flushCache);
    south.setBounds(left, rect.height - pt.y, rect.width, pt.y);
    bottom -= pt.y;
}
if (east != null) {
    Point pt = getSize(east, flushCache);
    east.setBounds(rect.width - pt.x, top, pt.x, (bottom - top));
    right -= pt.x;
}
if (west != null) {
    Point pt = getSize(west, flushCache);
    west.setBounds(left, top, pt.x, (bottom - top));
    left += pt.x;
}
if (center != null) {
    center.setBounds(left, top, (right - left), (bottom - top));
}
}

// 计算某一控件当前的大小, 长和宽
protected Point getSize(Control control, boolean flushCache) {
    return control.computeSize(SWT.DEFAULT, SWT.DEFAULT, flushCache);
}

// 设置该类中每个位置控件的属性的方法
protected void getControls(Composite composite) {
    // 获得当前面板中所有的控件对象
    Control[] children = composite.getChildren();
    // 循环所有控件, 并将每个控件所放的位置对号入座
    for (int i = 0, n = children.length; i < n; i++) {
        Control child = children[i];
        BorderData borderData = (BorderData) child.getLayoutData();
        if (borderData.region == SWT.TOP)
            north = child;
        else if (borderData.region == SWT.BOTTOM)
            south = child;
        else if (borderData.region == SWT.RIGHT)
            east = child;
    }
}

```



```
        else if (borderData.region == SWT.LEFT)
            west = child;
        else
            center = child;
    }
}
```

(3) 最后创建一个测试类，来看一下 BorderLayout 的最终实现。

TestBorderLayout.java

```
package com.fengmanfei.ch7;

import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.widgets.Text;

public class TestBorderLayout {
    public static void main(String[] args) {
        Display display = new Display();
        Shell shell = new Shell(display);
        shell.setSize(200, 150);
        shell.setLayout(new BorderLayout());

        Button buttonWest = new Button(shell, SWT.PUSH);
        buttonWest.setText("左");
        buttonWest.setLayoutData(new BorderData(SWT.LEFT));

        Button buttonEast = new Button(shell, SWT.PUSH);
        buttonEast.setText("右");
        buttonEast.setLayoutData(new BorderData(SWT.RIGHT));

        Button buttonNorth = new Button(shell, SWT.PUSH);
        buttonNorth.setText("上");
        buttonNorth.setLayoutData(new BorderData(SWT.TOP));

        Button buttonSouth = new Button(shell, SWT.PUSH);
        buttonSouth.setText("下");
        buttonSouth.setLayoutData(new BorderData(SWT.BOTTOM));

        Text text = new Text(shell, SWT.MULTI);
        text.setText("中间");
        text.setLayoutData(new BorderData());

        shell.pack();
        shell.open();
    }
}
```



```

while (!shell.isDisposed()) {
    if (!display.readAndDispatch()) {
        display.sleep();
    }
}
display.dispose();
}
}

```

这样，运行该测试类后，运行的效果如图 7.41 所示。

7.8 使用 VE 可视化布局

了解了这么多布局，都是使用手工编写代码的方式，那可不可以使用开发工具来设置控件的布局结构呢？在第 2 章中，已经安装了 Visual Editor 可视化的插件包。下面就看一下如何使用 VE 进行可视化的布局。

7.8.1 创建可视化的类

(1) 选择“文件”|“新建”|Visual Class 命令，在弹出的 New Java Visual Class 对话框中输入类名为 VESample，并设置 Style 为 Shell，如图 7.43 所示。单击“完成”按钮。

(2) 这样就利用向导创建了一个 Shell 窗口，如图 7.44 所示。首先熟悉一下可视化的编辑环境。

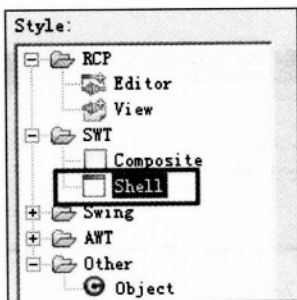


图 7.43 选择 Shell 选项

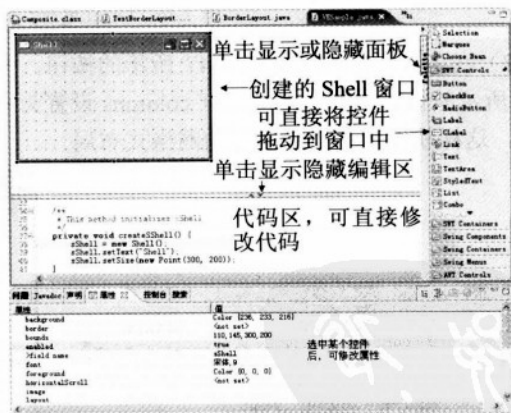


图 7.44 VE 编辑环境

- ❑ 右侧面板有各种各样的控件，当需要创建新控件时，只需要选中该控件，然后将其拖动到面板上所放置的位置即可。
- ❑ 各控件的属性均显示在“属性”视图中。要修改控件的属性，首先要选中该控件，然后在“属性”视图找到相关的属性，然后进行设置即可。

注意：此时若未出现图 7.44 所示的编辑环境，请确保该 .java 文件是在 Visual Editor 中打开的。在文件上右击，在弹出的快捷菜单中选择“打开方式”|Visual Editor 命令，这样打开文件就是以 VE 打开的了，如图 7.45 所示。

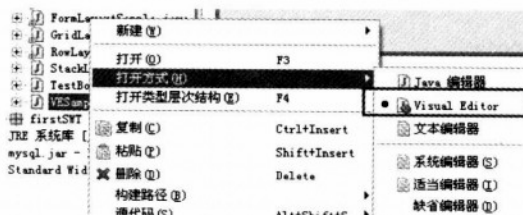


图 7.45 使用 VE 打开文件

7.8.2 进行布局设置

下面以 GridLayout 布局为例，介绍如何使用 VE 进行布局。

(1) 设置 Shell 窗口的布局为 GridLayout。选中 Shell 窗口，在属性面板中找到 layout 属性，选择 GridLayout，这样就设置了该面板的布局为 GridLayout。展开属性按钮，还可以查看具体的 Layout 设置。如图 7.46 所示为设置 GridLayout 属性。

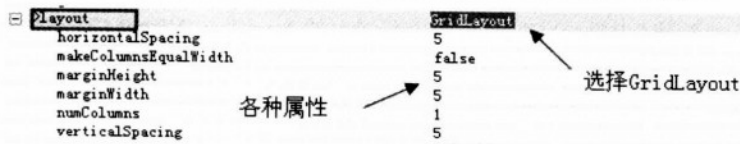


图 7.46 GridLayout 属性面板

(2) 在工具栏中单击如图 7.47 所示的按钮，弹出 Customize Layout 对话框，进行如图 7.48 所示的设置。将 Number of columns 设置为 4，并且选中 Make columns equal width 复选框，这样就设置了一个 4 列的网格式布局。

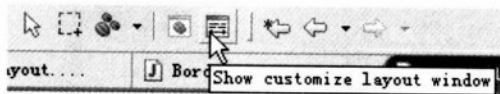


图 7.47 布局工具栏

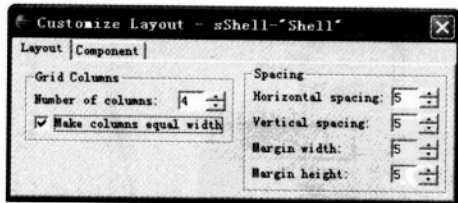


图 7.48 设置面板布局属性

(3) 添加所需要的控件。在控件面板上选中所需要的控件，然后放置到窗口面板的适当位置，如图 7.49 所示，此时已经添加了 3 个 Label 和两个文本框。

(4) 对控件作具体布局设置。下面要对其中一个文本框进行布局，选中这个文本控件，并单击图 7.47 所示的工具栏按钮，在弹出的对话框中选择 Component 选项卡，在这里可以对这个文本控件进行详细的设置。修改布局后的设置如图 7.50 所示，同时也可以看到设置

后的预览效果。

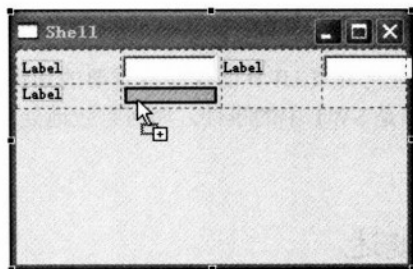


图 7.49 添加控件

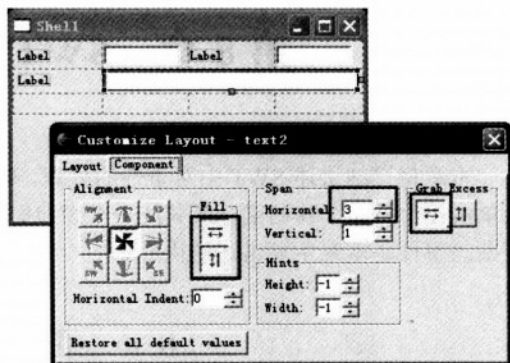


图 7.50 设置控件属性

(5) 同理，对其他的控件进行布局的设置。所有的控件设置完成后，最终的效果如图 7.51 所示。

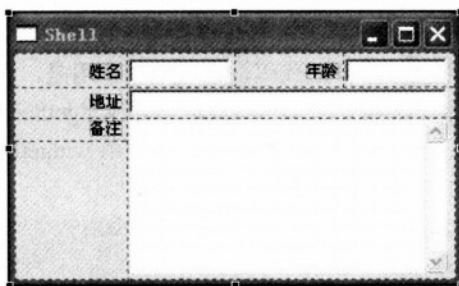


图 7.51 布局设计完成后的最终效果

综上所述，使用 Visual Editor 可视化的环境对布局进行设置能大大加快布局的设计。要想使用好编辑工具，也必须对布局的参数熟悉之后才能很快做出设计。所以即使有编辑器，请读者也要注意基本功的学习。

7.9 本章小结

通过本章的学习，读者应该对布局的原理有一个很深刻的理解了，布局是 SWT 中很重要的知识。另外，可视化开发带来的好处就是可以快速地进行布局，但也有弊端，由于代码是自动生成的，所以不易读懂，而且混乱，在进行再次开发时，不太方便。

第 8 章 SWT 中的事件模型

本章将详细讲述 SWT 中对事件的处理，事件的处理是开发 UI 界面程序很重要的知识。当单击一个按钮、移动鼠标或关闭一个窗口时，这些都是 SWT 中的事件。如何有效地处理复杂的事件是本章主要讲述的内容。

8.1 事件模型概述

SWT 中有关事件处理的类都打包在 `org.eclipse.swt.events` 中，该包中主要有一些监听器接口类，诸如类名为 `XXX Listener`，还有一些不同的事件类型，类名诸如 `XXXEvent`。此外还有一些适配器类，诸如 `XXXAdapter`。

在讲述如何处理事件之前，先学习一下与事件有关的常见的概念，有助于理解事件处理的机制。首先看一下以下处理事件的代码，该代码的作用是，有一个列表（List）对象，为该控件注册了事件，当某个选项处于选中状态时调用 `widgetSelected` 方法中的代码，输出 "widgetSelected 事件" 字符；当在某个选项上双击时，调用 `widgetDefaultSelected` 代码，输出 "widgetDefaultSelected 事件" 字符。

```
final List list = new List ( shell , SWT.NONE);
for ( int i=0;i<10;i++)
    list.add("Item"+i);
//事件处理的基本步骤
list.addSelectionListener( new SelectionListener(){
    public void widgetSelected(SelectionEvent e) {
        System.out.println("widgetSelected 事件");
    }
    public void widgetDefaultSelected(SelectionEvent e) {
        System.out.println("widgetDefaultSelected 事件");
    }
});
```

8.1.1 监听器（Listener）

在前面的代码中，创建了一个 `SelectionListener` 对象，`SelectionListener` 就是一个监听器对象。所谓监听器，也就是接收事件发生的对象，负责处理事件到达后响应事件的代码。监听器对象是一个接口，要响应一个事件就要实现该接口中的方法。监听器类的形式为 `XXXListener`，`XXX` 表示某一类型的监听器。例如，常用的监听器还有鼠标监听器（`MouseListener`）、键盘监听器（`KeyListener`）等。本例中使用的是选择监听器（`SelectionListener`），它是最常用的监

听器。

一个类型的监听器中有不同的响应事件的方法，例如本例中，`SelectionListener` 中要实现 `widgetSelected` 和 `widgetDefaultSelected` 两个方法。`widgetSelected` 表示选中事件，对于列表来说，就是单击列表中的一项所调用的方法。但对于按钮来说，就是单击这个按钮所调用的方法。不同的控件对事件的发布是不一样的。`widgetDefaultSelected` 事件对列表来说，就是双击列表中的一个事件所调用的方法。所以说同样是选中监听器，单击和双击所触发的事件是不一样的，因此要区别监听器中的方法是响应何种类型的事件。

8.1.2 事件 (Event)

当一个事件到达事件监听对象时，要携带一些该事件所附加的一些信息。比如，事件发生的时间，事件是由哪个控件发生的等。这些信息都是通过 `XXX Event` 形式来表示的，其中 `XXX` 表示事件的类型。例如本例中，选中事件所对应的是 `SelectionEvent` 对象，可以通过该对象获得发生该事件的一些信息，例如以下代码可以获得事件源控件：

```
public void widgetSelected(SelectionEvent e) {  
    List l = (List)e.widget;  
    System.out.println("widgetSelected 事件");  
}
```

注意，`e.widget` 获得控件对象需要进行强制类型转换。除了本例中使用的 `SelectionEvent` 外，`MouseListener` 对应的事件为 `MouseEvent`，键盘监听器对应的事件为 `KeyEvent` 等，每种事件携带的信息稍有不同，但不论是什么事件，都可以从事件中获得以下信息：

- ❑ `e.display`：事件发生时的 `Display` 对象。
- ❑ `e.data`：保存系统使用的数据，一般不常用。
- ❑ `e.widget`：事件发生的控件对象，通常要进行强制的类型转换，因为所有的控件都继承自 `Widget` 类。
- ❑ `e.time`：事件发生时的时间。

这是因为所有的 `XXXEvent` 类都继承自 `TypedEvent` 类。`TypedEvent` 类中的属性就是以上所列举的这些属性。对于具体的某一类型的事件，所携带类型的信息也稍有不同。

8.1.3 注册监听器

有了事件监听器和事件类型，还需要将该监听器对象注册给控件对象。注册是通过 `addXXXListener` 形式注册的，其中 `XXX` 为对应监听器对象的类型。例如本例中为列表控件注册选中事件的代码是 `list.addSelectionListener(...)`。当然，可以为一个控件注册多个事件监听器，不同的控件可以注册的监听事件是不同的。对 `Tree`（树控件）对象来说可以注册 `addTreeListener` 监听器，而其他的控件是没有该方法的。具体的控件可注册的监听器对象可参阅 SWT 的帮助文档。

⚠注意：为某一个控件注册对应的监听器后，也可以移除该监听器，需要使用 `removeXXXListener` 方法。例如，本例中使用 `removeSelectionListener(...)` 方法可以移除该控件所注册的监听器对象。

8.1.4 适配器

适配器可以简化事件处理的代码，形如 `XXXAdapter` 的类都是适配器类。例如，以上处理的程序中，如果使用适配器对象后，代码如下：

```
list.addSelectionListener( new SelectionAdapter(){
    public void widgetSelected(SelectionEvent e) {
        System.out.println("widgetSelected 事件");
    }
});
```

使用适配器类后，通过覆盖父类中的方法也可以达到与实现 `Listener` 接口同样的效果。其实，道理很简单，适配器类只是实现了 `Listener` 接口的抽象类。如下所示的是 `SelectionListener` 所对应的 `SelectionAdapter` 类的源代码：

```
package org.eclipse.swt.events;
public abstract class SelectionAdapter implements SelectionListener {
    public void widgetSelected(SelectionEvent e) {}
    public void widgetDefaultSelected(SelectionEvent e) {}
}
```

这样就不难理解适配器类的原理了。当然并不是每个监听器都对应一个适配器类，只有监听器类中响应时间的方法有多个时，才有必要使用监听器类。监听器的实现是选择适配器类还是实现接口的方式要根据不同的情况而确定。综上所述，为控件响应事件的方法有以下几个步骤：

- (1) 创建监听器对象，实现监听器中的方法（`XXXListener`）。
- (2) 为控件注册监听器（`addXXXListener`）。
- (3) 控件与监听器之间的桥梁是事件对象（`XXXEvent`）。
- (4) 对于监听器的接口中有多个事件响应方法的，也可以使用相应的适配器（`XXXAdapter`），通过覆盖父类中的方法来创建监听器对象。其中 `XXX` 为不同的事件类型。

8.1.5 常见的事件

表 8.1 列举了 SWT 中监听器与监听器所对应的事件。

表 8.1 SWT 事件一览表

事件类型	监听器和适配器	监听器中的方法	说明	可注册的 GUI 控件
ArmEvent	ArmListener	widgetArmed()	菜单项被选中之前触发此事件	MenuItem

续表

事件类型	监听器和适配器	监听器中的方法	说明	可注册的 GUI 控件
ControlEvent	ControlListener ControlAdapter	controlMoved()	控件的位置改变时	Control, TableColumn, Tracker
		controlResized()	控件的大小改变时	
DisposeEvent	DisposeListener	widgetDisposed()	控件释放时	Widget
FocusEvent	FocusListener FocusAdapter	focusGained()	控件获得焦点时	Control
		focusLost()	控件失去焦点时	
HelpEvent	HelpListener	helpRequested()	按 F1 键获得帮助 时触发该事件	Control, Menu, MenuItem
KeyEvent	KeyListener KeyAdapter	keyPressed()	按键按下时	Control
		keyReleased()	按键释放时	
MenuEvent	MenuListener MenuAdapter	menuHidden()	显示菜单时	Menu
		menuShown()	隐藏菜单时	
ModifyEvent	ModifyListener	modifyText()	文本被修改时	CCombo, Combo, Text, StyledText
MouseEvent	MouseListener MouseAdapter	mouseDoubleClick()	鼠标双击时	Control
		mouseDown()	鼠标按下时	
		mouseUp()	鼠标抬起时	
MouseMoveEvent	MouseMoveListener	mouseMove()	鼠标移动时	Control
MouseTrackEvent	MouseTrackListener MouseTrackAdapter	mouseEnter()	鼠标进入到控件区 域时	Control
		mouseExit()	鼠标离开该控件区 域时	
		mouseHover()	鼠标在该控件的区 域时	
PaintEvent	PaintListener	paintControl()	绘制控件时	Control
SelectionEvent	SelectionListener SelectionAdapter	widgetSelected()	选中控件时, 事件 的发生根据不同的 系统而异	Button, CCombo, Combo, CoolItem, CTabFolder, List, MenuItem, Sash, Scale, ScrollBar, Slider, StyledText, TabFolder, Table, TableCursor, TableColumn, TableTree, Text, TreeToolItem,
		widgetDefaultSelected()	默认选中控件时, 事件的发生根据不 同的系统而异	
ShellEvent	ShellListener ShellAdapter	shellActivated()	窗口被激活时	Shell
		shellClosed()	窗口关闭时	

续表

事 件 类 型	监听器和适配器	监听器中的方法	说 明	可注册的 GUI 控件
ShellEvent	ShellListener ShellAdapter	shellDeactivated()	窗口变为非激活状态时	Shell
		shellDeiconified()	当窗口不是最小化时	
		shellIconified()	当窗口最小化时	
TraverseEvent	TraverseListener	keyTraversed()	按下 Tab 键切换时触发该事件	Control
TreeEvent	TreeListener TreeAdapter	treeCollapsed	折叠树节点时	Tree, TableTree
		treeExpanded	展开树节点时	
VerifyEvent	VerifyListener	verifyText	改变文本时触发该事件	Text, StyledText

8.2 事件处理的常用写法

处理事件的关键是要实现 Listener 中的方法。实现接口也有多种方法，下面就来总结不同的情况下采用不同的写法。

8.2.1 内部匿名类

内部匿名类是最简单的一种方法，适用于对简单的事件处理。例如，以下所示的处理事件的方法就是使用的内部匿名类。

```
list.addSelectionListener( new SelectionListener(){
//程序代码
});
```

这种写法最简单，缺点是不能够使代码重用。使用这种方法注册的监听器，一旦注册后就不易移除。所以要想解决这个问题，可以使用内部类的方法。

8.2.2 内部类

将 8.2.1 节的程序改为如下：

```
SelectionListener listener = new SelectionListener(){
    public void widgetSelected(SelectionEvent e) {
    }
    public void widgetDefaultSelected(SelectionEvent e) {
    }
};
```

```
list.addSelectionListener(listener);  
list.removeSelectionListener(listener);
```

使用了内部类后，不仅为控件注册监听器，也可在需要时移除该监听器，并且也可以为多个控件注册同一个监听器。例如，一个菜单项与按钮具有相同的事件处理的方法，代码如下：

```
SelectionListener listener = new SelectionListener(){//省略代码  
MenuItem menuItem = new MenuItem( menu ,SWT.PUSH);  
menuItem.addSelectionListener(listener);  
Button button = new Button( shell , SWT.NONE);  
button.addSelectionListener(listener);
```

另外，通过内部类也可以处理不同的事件。例如，有两个按钮，分别注册了同一个监听器对象，如何区分是哪一个按钮所触发的事件呢？如下代码可以实现以下功能。

```
final Button bt1 = new Button( shell , SWT.NONE);  
final Button bt2 = new Button( shell , SWT.NONE);  
SelectionListener listener = new SelectionListener(){  
    public void widgetSelected(SelectionEvent e) {  
        Widget w = e.widget;  
        if ( w == bt1){//如果是 bt1 按钮所触发的事件  
            //代码处理  
        }else if ( w==bt2){ //如果是 bt2 按钮所触发的事件  
            //代码处理  
        }  
    }  
    public void widgetDefaultSelected(SelectionEvent e) {  
    }  
};  
bt1.addSelectionListener(listener);  
bt2.addSelectionListener(listener);
```

这种做法的好处是，可以将所有的事件集中到一个方法中进行处理，有利于代码后期的修改与维护。注意要在内部类中访问 bt1 对象，可以将 bt1 的类型设置为 final 型或是 static 类型。

8.2.3 实现接口的类

监听器对象本质上是一个接口，就可以利用接口的一般方法来处理，例如，有一个类实现了 SelectionListener 接口。代码如下：

```
public class MyListener implements SelectionListener{  
    public void widgetSelected(SelectionEvent e) {}  
    public void widgetDefaultSelected(SelectionEvent e) {}  
}
```

注册该监听器的对象的代码如下：

```
Button button= new Button( shell , SWT.NONE);  
button.addSelectionListener(new MyListener());
```

内部类的方法适用于在一个类中处理各种事件，而在实际的项目中往往是将所有的事件集中到一个类中。这样其他类中也能够重用该事件处理的代码，所以这种方式在项目中是经常使用的。

8.2.4 继承的类的方法

对于有对应的适配器类，还可以通过继承适配器类，然后覆盖适配器类中的方法来实现。例如，以下代码可以达到 8.2.3 节中实现接口的类方法同样的效果。

```
public class MyAdapter extends SelectionAdapter{  
    public void widgetSelected(SelectionEvent e) {  
    }  
    public void widgetDefaultSelected(SelectionEvent e) {  
    }  
}
```

注册该监听器的对象的代码如下：

```
Button button= new Button( shell , SWT.NONE);  
button.addSelectionListener(new MyAdapter ());
```

该方法与 8.2.3 节的方法最主要的区别是，通过继承的方式确保该类没有继承自其他类。如果此时该类继承自其他的类，那么只能采用实现接口的方法，如果该类没有继承自其他类，那么两种方法都可以使用。

总的来说，实现接口的方法更具有通用性，但继承的方法有一定局限性，而继承的方法简化了代码量。到底使用哪种方法，还要依据具体的情况而定。

8.3 键盘事件

键盘事件 `KeyEvent` 是常用的事件，用户对键盘的操作都是键盘事件。但相对于选中事件来说，键盘事件更为复杂，因为用户对键盘的操作有不同的类型。最简单的情况是只按下一个按键，稍微复杂的情况是同时配合辅助键，这就需要识别键盘事件的来源。

8.3.1 键盘事件程序示例

首先以一个程序来简单说明如何使用键盘事件。该程序的功能是有一个按钮，通过按“↑”、“↓”、“←”、“→”键可以移动按钮的位置。程序运行后的效果如图 8.1 所示，移动按钮位置后的效果如图 8.2 所示。

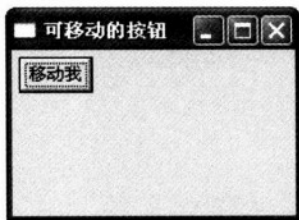


图 8.1 移动前的效果

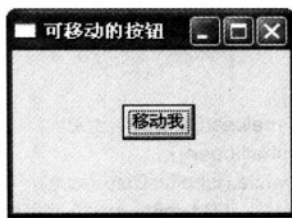


图 8.2 移动后的效果

该程序实现的代码如下：

MovingButton.java

```
package com.fengmanfei.ch8;

import org.eclipse.swt.SWT;
import org.eclipse.swt.events.KeyEvent;
import org.eclipse.swt.events.KeyListener;
import org.eclipse.swt.graphics.Rectangle;
import org.eclipse.swt.widgets.*;

public class MovingButton {

    public static void main(String[] args) {
        Display display = new Display();
        Shell shell = new Shell(display);
        shell.setText("可移动的按钮");
        Button button = new Button(shell, SWT.NONE);
        button.setBounds(5, 5, 50, 25);
        button.setText("移动我");
        //注册键盘事件监听器
        button.addKeyListener(new KeyListener() {
            //键盘按下后
            public void keyPressed(KeyEvent e) {
                //获得触发该事件的控件对象
                Control control = (Control)e.widget;
                //获得该控件的位置和大小
                Rectangle current = control.getBounds();
                if (e.keyCode == SWT.ARROW_DOWN)//如果按下了“下”按键
                    current.y++;//下移一个像素
                else if (e.keyCode == SWT.ARROW_UP)//如果按下了“上”按键
                    current.y--;//上移一个像素
                else if (e.keyCode == SWT.ARROW_LEFT)//如果按下了“左”按键
                    current.x--;//左移一个像素
                else if (e.keyCode == SWT.ARROW_RIGHT)//如果按下了“右”按键
                    current.x++;//右移一个像素
                //重新设置控件的位置
                control.setBounds(current);
            }
        })
    }
}
```



```

        public void keyReleased(KeyEvent e) {
        }

    });
    shell.setSize(200, 150);
    shell.open();
    while (!shell.isDisposed()) {
        if (!display.readAndDispatch())
            display.sleep();
    }
    display.dispose();
}
}

```

理解该程序的代码主要注意以下方面：

- ❑ 判断键盘按下的键是通过 `KeyEvent` 事件的 `keyCode` 属性来判断的，所以一定要弄清楚键盘事件 `KeyEvent` 对象中的各种属性的意义，这些属性的意义及其用法将在 8.3.2 节详细讲述。
- ❑ 获得事件发生的控件的位置和大小可以通过 `getBounds` 方法获得，而设置控件显示的位置可以通过 `setBounds` 方法获得。

8.3.2 键盘事件的各种属性

要获得事件源的信息，需要通过 `KeyEvent` 对象来获得。`KeyEvent` 除了 4 个基本的属性（见 8.1.1 小节）外，还有以下属性：

- ❑ `e.character`：表示输入字符的 Unicode 编码。如果输入的不是字符，如 Tab 键或 F1 等帮助键时，则该值为 `"\0"`。例如，`\u0041` 表示字符 A。也可以直接使用 'A' 进行判断。例如，以下代码可以判断按下的键是否是字符 A。

```

if(e.character=='A')
    System.out.println("用户按下了 A 键");

```

- ❑ `e.keyCode`：按键所对应的 ASCII 代码，键的常量封装在 `SWT` 类中。表 8.2 列举了按键与其对应的 `keyCode` 常量。

表 8.2 按键与其对应的常量表

按 键	keyCode 常量
方向键下	SWT.ARROW_DOWN
方向键上	SWT.ARROW_UP
方向键左	SWT.ARROW_LEFT
方向键右	SWT.ARROW_RIGHT
Alt 键	SWT.ALT
空格键	SWT.BS
Enter 键	SWT.CR

续表

按 键	keyCode 常量
Ctrl 键	SWT.CTRL
End 键	SWT.END
Esc 键	SWT.ESC
F1~F12 键	SWT.F1~SWT.F12
Home 键	SWT.HOME
Insert 键	SWT.INSERT
Page Down 下翻页键	SWT.PAGE_DOWN
Page UP 上翻页键	SWT.PAGE_UP
Shift 键	SWT.SHIFT
Tab 键	SWT.TAB
换行键	SWT.LF

- ❑ **e.stateMask**: 按下其他键的同时所使用的辅助键。辅助键包括 Ctrl 键、Shift 键和 Alt 键。判断键盘事件是否使用了辅助键的方法代码如下:

```
int bits = SWT.CTRL | SWT.ALT | SWT.SHIFT;
if ((e.stateMask & bits) == 0) {
    System.out.println("没有使用辅助键");
}
```

为了改进 8.3.1 节示例中移动按钮的程序, 现在增加一项功能。如果在按下按钮移动按钮的过程中使用了辅助键, 则不移动按钮, 这就需要增加一段判断是否使用了辅助键的代码。增加判断后的代码如下:

```
public void keyPressed(KeyEvent e) {
    int bits = SWT.CTRL | SWT.ALT | SWT.SHIFT;
    if ((e.stateMask & bits) != 0) {
        e.doit = false ;//取消该事件
        return;//并返回
    }
    //以下代码省略
}
```

判断是否使用了辅助键时使用了位运算符的方法, 如果读者有疑问, 请参阅 Java 的基本运算符相关的知识。

⚠注意: 不同的操作系统使用的辅助键可能不同。

- ❑ **e.doit**: 是否继续执行此事件。设置为 false, 则取消该事件。

为了更深刻地理解这些键盘事件各属性的意义, 将各个键盘事件所携带的信息进行比较, 如表 8.3 所示。

表 8.3 KeyEvent 事件比较

示 例 描 述	各属性的值	说 明
按下 A 键	character='a' keyCode=97 stateMask=0	97 表示键盘上的 a 键。当要输入大写字母 A 时，需要使用 Shift 键，所以这时 stateMask 的值为 SWT.SHIFT
按下 A 键同时按下 Shift 键	character='A' keyCode=97 stateMask= SWT.SHIFT	
按下 F1 键	character='\0' keyCode= SWT.F1 stateMask=0	F1 键不表示任何字符，所以 character 为 \0
按下 F1 键同时按下 Ctrl 键	character='\0' keyCode= SWT.F1 stateMask= SWT.CTRL	同时按下了辅助键，所以 stateMask 值为 SWT.CTRL

综上所述，若要判断输入的文本，则使用 character 属性；若要屏蔽一些快捷键的功能，则需要使用 keyCode 和 stateMask 属性。

8.4 鼠 标 事 件

鼠标事件也是很常用的事件，所以有必要对鼠标事件进行详细的讲解。涉及鼠标事件的事件类型有鼠标单击和双击事件（MouseEvent）、鼠标移动事件（MouseMoveEvent）、鼠标跟踪事件（MouseTrackEvent）。

8.4.1 鼠标事件程序示例

首先以一个程序为例来认识一下如何处理鼠标事件。该程序的功能是，对一个按钮实现拖动的功能，按住此按钮后可以将该按钮拖动到其他位置，释放按钮后停止拖动。程序运行后的效果如图 8.3 和图 8.4 所示。



图 8.3 拖放前的效果图

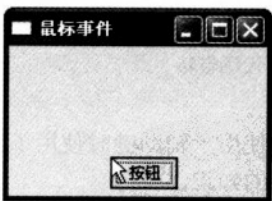


图 8.4 拖动后的效果图

该程序具体实现的代码如下：

MouseEventSample.java

```
package com.fengmanfei.ch8;

import org.eclipse.swt.SWT;
```

```
import org.eclipse.swt.events.*;
import org.eclipse.swt.graphics.Point;
import org.eclipse.swt.graphics.Rectangle;
import org.eclipse.swt.widgets.*;

public class MouseEventSample {
    private Shell shell;
    private Button button;
    private MoveButtonListener listener;
    public MouseEventSample() {
        shell = new Shell();
        shell.setText("鼠标事件");
        button = new Button(shell, SWT.NONE);
        button.setBounds(5, 5, 50, 25);
        button.setText("按钮");
        listener = new MoveButtonListener();//创建监听器对象
        button.addMouseListener(listener);//注册鼠标监听器
        button.addMouseTrackListener(listener);//注册鼠标跟踪监听器
        shell.setSize(200, 150);
    }
    //监听器类，实现了鼠标监听器的接口
    public class MoveButtonListener implements MouseMoveListener, MouseListener, Mouse
    TrackListener{
        //当鼠标移动时
        public void mouseMove(MouseEvent e) {
            //首先当前鼠标的位置转化为窗口的坐标位置
            Point convertPoint = Display.getCurrent().map(button, shell, e.x,e.y);
            Rectangle current = button.getBounds();
            //重新设置按钮的位置，使之跟随鼠标移动
            button.setBounds(convertPoint.x, convertPoint.y, current.width, current.height);
        }
        //鼠标双击事件
        public void mouseDoubleClick(MouseEvent e) {
            System.out.println("双击了该按钮");
        }
        //当鼠标按下时，为按钮注册鼠标移动监听器
        public void mouseDown(MouseEvent e) {
            button.addMouseListener(listener);
        }
        //当鼠标抬起时，停止拖放，移除鼠标移动监听器
        public void mouseUp(MouseEvent e) {
            button.removeMouseListener(listener);
        }
        //当鼠标进入到按钮区域时
        public void mouseEnter(MouseEvent e) {
            System.out.println("鼠标进入到按钮区域");
        }
        //当鼠标离开按钮区域时
```

```
        public void mouseExit(MouseEvent e) {
            System.out.println("鼠标离开按钮区域");
        }
        //当鼠标在按钮区域时
        public void mouseHover(MouseEvent e) {
            System.out.println("鼠标在按钮区域");
        }
    }
    public Button getButton() {
        return button;
    }
    public void setButton(Button button) {
        this.button = button;
    }
}

public Shell getShell() {
    return shell;
}
public void setShell(Shell shell) {
    this.shell = shell;
}
}
public static void main(String[] args) {
    Display display = Display.getDefault();
    MouseEventSample sample = new MouseEventSample();
    sample.getShell().open();
    while (!sample.getShell().isDisposed()) {
        if (!display.readAndDispatch())
            display.sleep();
    }
    display.dispose();
}
}
```

实现此拖动按钮程序的设计思路有以下几点:

- ❑ 拖动过程中, 按钮跟随鼠标的效果是通过在鼠标移动事件 `mouseMove` 的方法中, 设置按钮的位置为鼠标移动事件所发生的位置。`mouseMove` 方法为 `MouseMoveListener` 监听器中的方法。
- ❑ `MouseMoveListener` 监听器对象并不是程序运行时就注册给按钮, 而是在按钮上单击时, 才注册该监听器。在按钮抬起时, 则移除该监听器。
- ❑ 通过 `MouseEvent` 对象获得的事件发生的 `x`、`y` 坐标是相对于按钮的, 此时要设置按钮的位置, 要转换成相对于窗口的坐标位置, 这是通过 `Display` 对象的 `map` 方法计算的。
- ❑ 不同的鼠标事件可能会有同样的方法, 如发生鼠标移动时, `mouseMove` 也会发生鼠标在该按钮区域事件 `mouseHover`。所以要仔细区分每种不同鼠标事件所对应的事件处理。

8.4.2 鼠标事件的各种属性

无论是哪种类型的鼠标监听器，当鼠标事件发生时，都是通过 `MouseEvent` 来传递事件发生时所携带的信息的。下面就来看一下 `MouseEvent` 的一些属性。

- ❑ `e.button`: 鼠标按键的 ID，从左向右依次为 0、1、2。如果设置的鼠标为左手使用，则从右向左排列，该属性一般用的机会很少。
- ❑ `e.stateMask`: 在使用鼠标时同时按下的辅助键，与键盘使用的辅助键相同。使用常量 `SWT.BUTTON_MASK` 可以判断是否使用了辅助键。例如以下代码：

```
if ((e.stateMask & SWT.BUTTON_MASK) == 0) {  
    System.out.println("没有按下任何辅助键");  
}
```

- ❑ `e.x`: 事件发生时，鼠标相对于该事件发生的控件的 x 坐标。例如，本例中监听器注册的是按钮控件，不是窗口控件，所以获得的事件发生时，鼠标相对于按钮的 x 坐标。
- ❑ `e.y`: 事件发生时，鼠标相对于该事件发生的控件的 y 坐标。事件发生的坐标属性是常用的属性。

8.5 其他常用的事件

除了键盘和鼠标事件外，还有其他的一些事件比较常用。本节将介绍选中事件（`SelectionEvent`）和文本修改事件（`VerifyEvent` 和 `ModifyEvent`）。

8.5.1 选中事件

大多数控件都支持选中事件（`SelectionEvent`），但对于不同的控件，触发选中事件的情况是不同的。例如，按钮注册了选中事件监听器，则单击按钮时触发该事件，若为菜单项注册了选中事件监听器，则在单击菜单项时触发该事件。

在 8.1.1 节中的示例程序就是使用选中事件来处理的，这里就不多举例子了。下面具体了解一下 `SelectionEvent` 的各种属性：

- ❑ `e.item`: 选中事件发生时，一般可以进行强制类型转换获得触发该事件的控件。例如，以下代码为获得事件发生的控件的方法，假设该控件为按钮。

```
public void widgetSelected(SelectionEvent e) {  
    Button bt = (Button) e.item; // 因为 Widget 是 Button 的父类  
}
```

- ❑ `e.detail`: 事件发生时附加的额外信息，根据不同的控件，有不同的值。表 8.4 列举了不同的控件触发该事件时所携带的不同的值。

表 8.4 不同控件所对应的常量

控 件	对应的常量
Sash	SWT.DRAG
ScrollBar 和 Slider	SWT.DRAG、SWT.HOME、SWT.END、SWT.ARROW_DOWN、SWT.ARROW_UP、SWT.PAGE_DOWN、SWT.PAGE_UP
Table and Tree	SWT.CHECK
CoolItem and ToolItem	SWT.ARROW

获得该属性时，只能是表 8.4 所示的这些控件触发选中事件时才携带所对应的常量值，若为其他的控件，该值为 null。

- ☐ e.x、e.y、e.width 和 e.height：分别为事件发生的 x、y 坐标，宽和高。不同的控件触发该事件，会有不同的值。该属性一般不常用。
- ☐ e.stateMask：选中控件时所使用的辅助键。与键盘事件中的 stateMask 属性意义相同。
- ☐ e.text：针对带有超级链接的 Link 控件所使用的，值为超级链接中 href 的值。有关的示例程序请参阅 9.1 节。
- ☐ e.doit：是否执行该事件。

8.5.2 文本修改程序示例

涉及文本修改事件，有两个事件——VerifyEvent 和 ModifyEvent。VerifyEvent 事件比 ModifyEvent 事件所携带的事件信息多，而且 VerifyEvent 在 ModifyEvent 之前发生。下面以一个程序来说明如何使用 VerifyEvent 和 ModifyEvent。

该程序的功能是，有两个文本框，上边的文本框只能输入小写字母，下边的文本框只能输入大写字母。当对上边的文本框进行输入时，下边的文本框也同时显示自动输入上边文本框的值。该程序运行后的效果如图 8.5 所示。

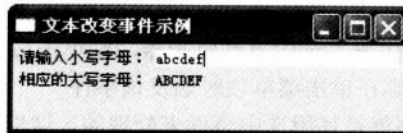


图 8.5 程序运行后的效果图

该程序的具体实现代码如下：

ModifyTextEvent.java

```
package com.fengmanfei.ch8;

import org.eclipse.swt.SWT;
import org.eclipse.swt.events.VerifyEvent;
import org.eclipse.swt.events.VerifyListener;
import org.eclipse.swt.layout.GridData;
```



```
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.*;

public class ModifyTextEvent implements VerifyListener{

    private Shell shell;
    private Text top;
    private Text bottom;
    public ModifyTextEvent(){
        shell = new Shell();
        shell.setText("文本改变事件示例");
        shell.setLayout( new GridLayout(2,false));
        new Label( shell, SWT.NONE).setText("请输入小写字母: ");
        top = new Text(shell, SWT.SINGLE);
        top.setLayoutData( new GridData(GridData.FILL_HORIZONTAL));
        top.addVerifyListener( this );
        new Label( shell, SWT.NONE).setText("相应的大写字母: ");
        bottom = new Text(shell, SWT.SINGLE);
        bottom.setLayoutData( new GridData(GridData.FILL_HORIZONTAL));
        bottom.setEditable(false);
        shell.setSize(300,100);
    }
    public Shell getShell() {
        return shell;
    }
    public void setShell(Shell shell) {
        this.shell = shell;
    }
    //VerifyListener 接口中的方法
    public void verifyText(VerifyEvent e) {
        char c = e.character;//获得输入的字符
        //如果该字符不为字母或者不是大写字母，取消输入
        if (!Character.isLetter(c)||Character.isUpperCase(c)){
            e.doit=false;
            return;
        }
        //在下边的文本框中输入相对应的大写字母
        bottom.append(" "+Character.toUpperCase(c));
    }
    public static void main(String[] args) {
        Display display =Display.getDefault();
        ModifyTextEvent sample = new ModifyTextEvent();
        sample.getShell().open();
        while (!sample.getShell().isDisposed()) {
            if (!display.readAndDispatch())
                display.sleep();
        }
        display.dispose();
    }
}
```

要获得 VerifyEvent 事件发生时所输入的字符，可以使用属性 e.character。如果输入的不是小写字母，则将该事件的 e.doit 属性设置为 false，不会输入该字符。VerifyEvent 常用来判断输入字符的有效性。

8.5.3 文本修改事件：VerifyEvent 的各种属性

VerifyEvent 类从继承关系上来说继承自 KeyEvent，所以具有 KeyEvent 的属性。如图 8.6 所示为 VerifyEvent 类的继承关系示意图。

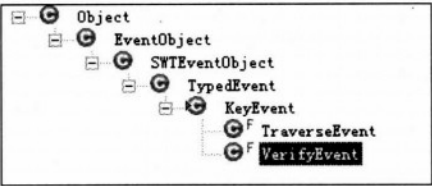


图 8.6 VerifyEvent 类的继承关系示意图

VerifyEvent 对象除了携带 KeyEvent 属性外还有其他的一些属性：

- ❑ e.start 和 e.end：修改字符的起始位置。
- ❑ e.text：新插入的文本。

为了弄清楚不同的属性所代表的意义，表 8.5 对常见的修改文本事件和事件所携带的信息进行了比较。

表 8.5 VerifyEvent 事件比较

文本框的初始值	示例描述	改变字符后文本框的值	各属性的值	说明
"abcd"	字符后输入字符 e	"abcde"	character='e' keyCode=101 stateMask=0 doit=true start=4 end=4 text=e	插入字符时 start 和 end 的值相等，记录的都是插入字符所在的位置
"abcd"	从剪贴板中粘贴 "abcd" 字符	"abcdabcd"	character='\0' keyCode=0 stateMask=0 doit=true start=4 end=4 text=abcd	
"abcd"	删除字符 c	"abd"	character=' ' keyCode=8 stateMask=0 doit=true start=2 end=3 text=	当删除字符时，text 的值为空，start 记录了删除字符的起始位置，end 记录了删除字符的结束位置
"abcd"	此时选中 cd 字符后按 Delete 键	"ab"	character=' ' keyCode=8 stateMask=0 doit=true start=2 end=4 text=	

下面以一小程序来判断用户进行的操作是插入还是删除操作：

```
final Text text = new Text(shell, SWT.SINGLE);
text.addVerifyListener( new VerifyListener(){
    public void verifyText(VerifyEvent e) {
        int start =e.start;
        int end = e.end;
        if (start==end)//表示为插入操作
```

```
System.out.println("您在"+e.start+"位置, 插入了"+e.text+"字符");  
else//表示为删除操作  
    System.out.println("您删除了字符"+ text.getText().substring(e.start,e.end));  
}  
});
```

这里判断是插入还是删除操作的方法是通过比较 `e.start` 和 `e.end` 的值来判断的, 当两个值相等时, 表示为插入操作, 此时通过 `e.text` 可以获得插入的文本。当两个值不相等时, 则表示为删除操作, 但删除的文本不能直接获得, 需要间接方法才能获得。

8.5.4 文本修改事件: `VerifyEvent` 和 `ModifyEvent` 的区别

同样是文本修改事件, `VerifyEvent` 和 `ModifyEvent` 之间主要有以下方面的区别:

1. `VerifyEvent` 比 `ModifyEvent` 携带更多的事件信息

通过以上的程序可以知道 `VerifyEvent` 可以携带更多事件信息, 信息包括修改的字符和字符所在的位置。而 `ModifyEvent` 则不能够获得这些信息, 它只知道文本是否被修改了, 具体修改了哪些内容, 且不能获得。因此, `ModifyEvent` 事件一般用于保存文本等操作, 这些操作不需要获得修改文本的详细信息。`VerifyEvent` 事件适用于对修改文本有效性的判断, 因为此操作可能需要查询修改的具体信息。

2. `VerifyEvent` 事件发生的优先级高于 `ModifyEvent`

`VerifyEvent` 总是优先于 `ModifyEvent` 事件的发生。当文本修改时, 总是首先触发 `VerifyEvent` 事件, 当该事件的 `doit` 属性为 `true` 时, 才触发 `ModifyEvent` 事件。也就是说当文本修改时, `VerifyEvent` 事件总是会发生的, 但只有 `VerifyEvent` 事件的 `doit` 属性为 `true` 时, 才会触发 `ModifyEvent` 事件。这两种事件发生的流程图如图 8.7 所示。

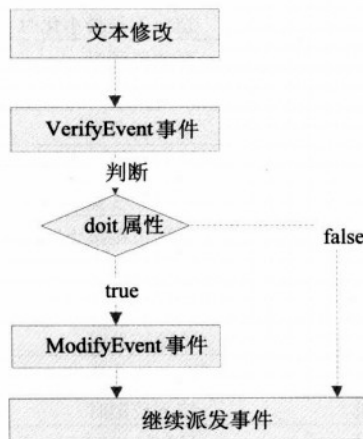


图 8.7 `VerifyEvent` 事件与 `ModifyEvent` 事件发生的流程图

因此, 要跟据不同的情况来选择是使用 `VerifyEvent` 还是 `ModifyEvent` 事件。

8.6 无类型的事件

之前学习的事件类型都是有类型的事件，即以上的事件都是继承自 `TypedEvent` 类。有类型的事件更加面向对象，但效率比较低。使用无类型的事件虽然效率比较高，但是用无类型事件处理代码会比较繁琐。虽然两种类型的事件所达到的效果相同，但各有优点。

8.6.1 注册无类型事件监听器

注册一个无类型事件的监听器对象，需使用以下方法：

```
addListener (int eventType, Listener listener)
```

其中，`eventType` 为 SWT 中封装的常量，表示事件的类型，这些常量如表 8.6 所示。`listener` 为监听器对象，该监听器接口中只有一个方法 `void handleEvent (Event event)`。监听器对象要实践这个方法。

表 8.6 SWT 类所代表的事件常量

事件类型常量	说 明
SWT.Activate	当激活窗口时
SWT.Arm	菜单项被选中之前
SWT.Close	当关闭窗口时
SWT.Collapse	折叠树的节点时
SWT.Deactivate	窗口处于非激活状态时
SWT.DefaultSelection	默认选中时
SWT.Deiconify	窗口不是最小化时
SWT.Dispose	释放资源时
SWT.DragDetect	拖动控件时
SWT.Expand	展开树节点时
SWT.FocusIn	控件获得焦点时
SWT.FocusOut	控件失去焦点时
SWT.HardKeyDown	硬件按键按下时，例如 Pocket PC
SWT.HardKeyUp	硬件按键抬起时，例如 Pocket PC
SWT.Help	按下帮助键时
SWT.Hide	隐藏控件时
SWT.Iconify	窗口最小化时
SWT.KeyDown	按下按键时
SWT.KeyUp	抬起按钮时
SWT.MenuDetect	选中菜单时
SWT.Modify	文本修改时
SWT.MouseDoubleClick	双击鼠标时

续表

事件类型常量	说 明
SWT.MouseDown	鼠标按下时
SWT.MouseEnter	鼠标进入时
SWT.MouseExit	鼠标离开时
SWT.MouseHover	鼠标在控件区域上时
SWT.MouseMove	鼠标移动时
SWT.MouseUp	鼠标抬起时
SWT.Move	移动控件时
SWT.None	无类型事件
SWT.Paint	绘制控件时
SWT.Resize	重新设置控件大小时
SWT.Selection	选中控件时
SWT.Show	显示控件时
SWT.Traverse	使用 Tab 键切换时
SWT.Verify	文本进行修改时

8.6.2 无类型事件程序示例

下面以一个程序示例来说明如何使用无类型的事件。在 8.3.1 节中，编写过一个移动按钮的程序，现在将该程序改造一下，看一下如何使用无类型的事件也达到同样的功能效果。修改成无类型事件后的程序代码的主要部分如下：

UntypedEvent.java

```

Button button = new Button(shell, SWT.NONE);
button.setBounds(5, 5, 50, 25);
button.setText("移动我");
Listener listener = new Listener() {
    public void handleEvent(Event e) {
        if (e.type == SWT.KeyDown) { //如果是按下按键事件
            Control control = (Control) e.widget;
            // 获得该控件的位置和大小
            Rectangle current = control.getBounds();
            if (e.keyCode == SWT.ARROW_DOWN) // 如果按下了“下”按键
                current.y++; // 下移一个像素
            else if (e.keyCode == SWT.ARROW_UP) // 如果按下了“上”按键
                current.y--; // 上移一个像素
            else if (e.keyCode == SWT.ARROW_LEFT) // 如果按下了“左”按键
                current.x--; // 左移一个像素
            else if (e.keyCode == SWT.ARROW_RIGHT) // 如果按下了“右”按键
                current.x++; // 右移一个像素
            // 重新设置控件的位置
            control.setBounds(current);
        } else if (e.type == SWT.Selection) { //如果是选中事件

```

```
        System.out.println("你单击了此按钮");
    }
}
};
// 注册无类型的事件监听器
button.addListener(SWT.KeyDown, listener);
button.addListener(SWT.Selection, listener);
```

通过以上程序可以总结出使用无类型事件时所应该注意的问题:

- ☐ 要注册无类型事件的监听器, 必须实现 `Listener` 接口, 该接口中有一个方法 `handleEvent`, 该方法为事件发生时所调用的方法。
- ☐ 通过 `e.type` 可以判断事件发生的类型。
- ☐ 在注册监听器时要同时指定该监听器的事件的类型。

综上所述, 无类型的事件是最底层的事件。而为了更加面向对象, 使代码变得简洁, 所以开发了有类型的事件。简单地说, 有类型的事件是将无类型的事件进行包装。无论哪种类型的事件, 都会调用 `handleEvent` 方法, 只是有类型的事件将该方法作了判断处理。有兴趣的读者可以阅读 `org.eclipse.swt.widgets.TypedListener` 类中的源代码部分。

虽然有无类型的事件, 但还是推荐读者使用有类型的事件, 毕竟这样的代码结构更符合面向对象的思想。

8.7 本章小结

通过本章的学习, 读者对 SWT 事件的处理有详细的了解。如果读者之前使用过 Swing 来进行 UI 编程, 那么相对于这部分而言就相对容易, 因为 SWT 的事件处理机制与 Swing 的事件处理机制差不多。

如何处理事件很简单, 只要实现这些事件提供的监听器接口即可。但是笔者希望读者也能够了解一些事件处理模型的设计模式, 这样也有助于学习其他事件处理模型。事实上, SWT 事件处理模式也采用了“观察者 (Observer)”的设计模式, 希望读者能够参阅一下这方面的资料, 更深刻地理解事件处理的模式。



第 3 篇



DESIGN

SWT 高级篇

第 9 章 SWT 高级控件

第 10 章 SWT 中的拖放支持

第 11 章 SWT 线程

第 12 章 SWT 系统资源

第 13 章 SWT 的高级应用

设计手册

PDG

第9章 SWT 高级控件

本章将详细介绍 SWT 的高级控件，包括菜单、工具栏、可拖动的工具栏、系统托盘、滑动组件、对话框、表格、树和浏览器等。通过本章的学习，您对 SWT 中常用的控件将有一个全面的了解。

9.1 链接文本 (Link)

链接文本是可以像网页中带有超级链接的文本。如图 9.1 所示为链接文本的示意图。当鼠标移动到链接文本上时，鼠标将会变成手状鼠标。

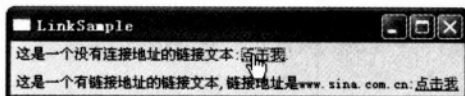


图 9.1 链接文本示意图

创建该程序的代码如下：

```
//创建链接文本对象
Link link = new Link(shell, SWT.NONE);
//设置链接文本中的字符，与网页中的超级链接文本标记类似
//<a href="链接地址">链接文本</a>
//其中\n 可以表示为换行符
String text = "这是一个没有连接地址的链接文本:<A>点击我</A>.\n\n 这是一个有链接地址的链接文  
本,链接地址是 www.sina.com.cn:<A href=\"www.sina.com.cn\">点击我</A>";
link.setText(text);
link.setSize(400, 400);
//注册点击文本事件
link.addListener (SWT.Selection, new Listener () {
    public void handleEvent(Event event) {
        //显示出打印的文本
        System.out.println("您点击的是: " + event.text);
    }
});
```

使用链接文本有以下几点需要注意：

- ❑ 链接文本只是将带有<A>标记的文本显示为链接的形式。
- ❑ 如果指定了 href 的地址，当通过事件调用 event.text 方法时，得到的是 href 中的地址。例如，在示例程序中，当点击点击我这个链接时，输出的是 www.sina.com.cn；当点击<A>点击我时，输出的是“点击我”。

- 在设置文本的字符串中，如果遇到“\n”标记，文本会自动换行。

9.2 菜单（Menu 和 MenuItem）

菜单（Menu）是应用程序中常见的控件，菜单有放置在窗口上方的系统菜单，也有单击鼠标右键时弹出的快捷菜单。如图 9.2 所示的就是一个典型的系统菜单。

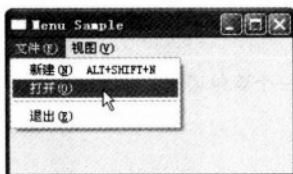


图 9.2 带有菜单的窗口

创建该系统菜单的代码如下：

MenuSample.java

```
final Shell shell = new Shell(display);
shell.setText("Menu Sample");
// 创建主菜单，用于放置到菜单条上
Menu main = new Menu(shell, SWT.BAR);

// 主菜单的第一个菜单项
MenuItem file = new MenuItem(main, SWT.CASCADE);
file.setText("文件(&F)");

// 文件菜单，为下拉式
Menu filemenu = new Menu(shell, SWT.DROP_DOWN);
// 新建菜单项
MenuItem newItem = new MenuItem(filemenu, SWT.PUSH);
newItem.setText("新建(&N) ALT+SHIFT+N");
// 设置快捷键
newItem.setAccelerator(SWT.ALT + SWT.SHIFT + 'N');
// 打开菜单项
MenuItem openItem = new MenuItem(filemenu, SWT.PUSH);
openItem.setText("打开(&O)");
// 分割线菜单项
new MenuItem(filemenu, SWT.SEPARATOR);
// 退出菜单项
MenuItem exitItem = new MenuItem(filemenu, SWT.PUSH);
exitItem.setText("退出(&E)");
// 为退出菜单项注册事件
exitItem.addListener(SWT.Selection, new Listener() {
    public void handleEvent(Event event) {
        // 当单击退出菜单时，释放窗口
        shell.dispose();
    }
});
```

```

    }
});
// 将文件菜单放置到主菜单的第一个菜单项上
file.setMenu(filemenu);

// 主菜单的第二个菜单项
MenuItem view = new MenuItem(main, SWT.CASCADE);
view.setText("视图(&V)");

Menu viewMenu = new Menu(shell, SWT.DROP_DOWN);
MenuItem normalItem = new MenuItem(viewMenu, SWT.RADIO);
normalItem.setText("普通(&N)");
// 将文件菜单放置到主菜单的第一个菜单项上
view.setMenu(viewMenu);
// 将主菜单放置到窗口上
shell.setMenuBar(main);

```

9.2.1 菜单与菜单项之间的关系

菜单的使用并不复杂，关键是要理解菜单和菜单项之间的关系。与前面学习的选项卡文件夹（TabFolder）和选项卡（TabItem）的关系类似，菜单（Menu）与菜单项（MenuItem）的关系也是一对多的关系。也就是说，一个菜单可以包含多个菜单项。那么，一个菜单是通过什么方法绑定到一个控件上的呢？

- 对于所有控件（Control）都可以使用 `setMenu(Menu menu)` 方法将一个菜单绑定到一个控件，这样绑定的菜单为右键弹出式菜单。
- 对于 Shell 窗口控件，除了可以使用 `setMenu(Menu menu)` 方法设置弹出式菜单外，还可以使用 `setMenuBar(Menu menu)` 设置显示在窗口上方的菜单。

下面就以刚才的程序为例，来理解菜单与菜单项的关系。如图 9.3 所示为上述程序中菜单与菜单项之间的关系。

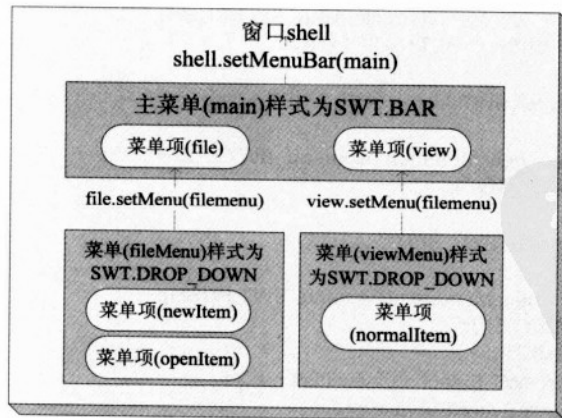


图 9.3 菜单（Menu）与菜单项（MenuItem）之间的关系

9.2.2 菜单的样式

菜单样式中必选其一的样式有 3 种

1. SWT.BAR: Shell 菜单栏中的菜单

例如, 上述程序中, 主菜单为一个 SWT.BAR 菜单。SWT.BAR 总是与 setMenuBar(Menu menu)方法一起使用。例如:

```
Menu main = new Menu(shell, SWT.BAR);
shell.setMenuBar(main);
```

若此时将创建菜单的样式改为其他的, 而不是 SWT.BAR, 运行时将会抛出“Menu is not a BAR”异常。

2. SWT.DROP_DOWN: 从菜单栏或菜单项弹出的菜单

例如, 上述程序中使用的“文件”菜单就是一个下拉式菜单。创建下拉式菜单时, 所放置这个菜单的菜单项样式必须是 SWT.CASCADE, 否则会抛出“Menu item is not a CASCADE”异常。创建弹出式菜单有以下两种方法。

方法 1:

```
MenuItem parentItem = new MenuItem(main, SWT.CASCADE); //弹出菜单的菜单项
Menu menu = new Menu(shell, SWT.DROP_DOWN); //弹出的菜单
parentItem.setMenu(menu); //将菜单附加到菜单项上
```

方法 2:

```
MenuItem parentItem = new MenuItem(main, SWT.CASCADE); //弹出菜单的菜单项
//如果将附加的菜单项作为构造方法, 默认为下拉式菜单
Menu menu = new Menu(parentItem);
parentItem.setMenu(menu); //将菜单附加到菜单项上
```

3. SWT.POP_UP: 单击鼠标右键弹出的菜单

这种弹出式菜单适用于作为上下文菜单。例如, 将上述程序中的代码修改如下:

```
//.....前面的代码相同, 代码中的第 3 行改为
Menu main = new Menu(shell, SWT.POP_UP);
//.....省略, 代码中的最后一行改为
shell.setMenu( main );
```

修改代码以后, 在窗口的任意一个位置单击鼠标右键, 将弹出右键菜单。程序运行的效果如图 9.4 所示。

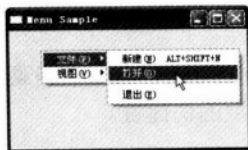


图 9.4 弹出式菜单

另外，菜单还有几个可选的样式。

- ❑ **SWT.NO_RADIO_GROUP**: 使单选按钮组的功能不生效，只针对菜单项的样式为 **SWT.RADIO** 时有效。
- ❑ **SWT.LEFT_TO_RIGHT** 或 **SWT.RIGHT_TO_LEFT**: 设置菜单显示的方式是由左向右还是由右向左，默认为由左向右。

9.2.3 菜单项的样式

下面介绍菜单项的样式。

- ❑ **SWT.PUSH**: 普通的菜单项，单击类似于按钮的操作。
- ❑ **SWT.RADIO**: 可以在几个选项之间选中一个，并且选中的选项以黑原点表示。
- ❑ **SWT.CHECK**: 类似于多选框，可以同时选中多个选项，并且选中的选项以对勾符号表示。
- ❑ **SWT.CASCADE**: 可以包含一个下拉菜单的菜单项。
- ❑ **SWT.SEPARATOR**: 充当分割条的菜单项，通常没有其他意义。

图 9.5 显示了这几种不同样式的菜单项。

使用菜单项时注意以下几点：

- ❑ 对使用 **SWT.RADIO** 和 **SWT.CHECK** 样式的菜单项，可以使用 `setSelection(boolean selected)` 方法来设置是否为选中状态。若为 `true`，则为选中状态；若为 `false`，则为未选中状态。
- ❑ 对使用 **SWT.RADIO** 的菜单项，如果这时菜单的样式设置为 **SWT.NO_RADIO_GROUP**，则可以同时选择多个选项。

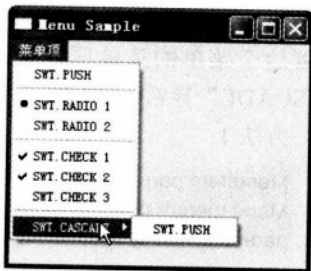


图 9.5 5 种不同样式的菜单项

9.2.4 设置菜单项的图标

通常，为了使界面更加友好，可以为菜单项设置图标。设置图标的方法是 `setImage(Image image)`。图 9.6 显示了设置图标后菜单项的外观。

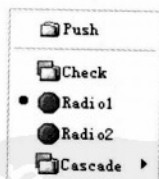


图 9.6 带有图标的菜单项

9.2.5 设置菜单项快捷键

通常，为了方便快速地使用某个功能，会为一些菜单项设置快捷键，这样，很大程度上提高了界面的友好性。有两种类型的快捷键。

1. 菜单栏中的快捷键：带下划线的快捷键

如图 9.7 所示，“文件”菜单项中有一个带下划线的字母“F”，说明按 Alt+F 组合键可以调用该菜单。这种类型的快捷键的设置代码如下：

```
MenuItem file = new MenuItem(main, SWT.CASCADE);
file.setText("文件(&F)");
```

这里是使用符号“&”加上字母来标识快捷键的。

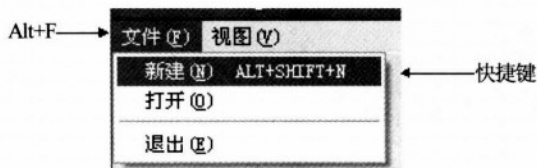


图 9.7 两种快捷键

2. 使用 setAccelerator (int accelerator)方法创建的快捷键

如图 9.7 所示，新建菜单也可以通过使用 Alt+Shift+N 组合键来调用，这是通过设置了快捷键的结果，例如图中的菜单项设置快捷键的代码如下：

```
MenuItem newItem = new MenuItem(filemenu, SWT.PUSH);
newItem.setText("新建(&N) ALT+SHIFT+N");
// 设置快捷键
newItem.setAccelerator(SWT.ALT + SWT.SHIFT + 'N');
```

设置的快捷键可以使用 SWT 中的常量，例如使用 F1 键对应 SWT.F1，若使用组合键可以用“+”将几个键组合。

9.3 工具栏 (ToolBar 和 ToolItem)

工具栏与菜单类似，也是一般桌面程序中常见的一种控件。如图 9.8 和图 9.9 所示为两种不同风格的工具栏。



图 9.8 带有图标和文字的工具栏



图 9.9 只带有图标的工具栏

创建如图 9.8 所示工具栏的代码如下：

ToolBarSample.java

```
package com.fengmanfei.ch9;

import org.eclipse.swt.SWT;
import org.eclipse.swt.graphics.*;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.widgets.*;
import com.fengmanfei.util.ImageFactory;

public class ToolBarSample {
    public static void main(String[] args) {
        Display display = new Display ();
        Shell shell = new Shell (display);
        shell.setText("ToolBar");
        shell.setLayout( new GridLayout());
        Composite tool = new Composite( shell ,SWT.NONE);
        tool.setLayoutData(new GridData(SWT.LEFT,SWT.TOP,true ,false));
        //创建工具栏对象, 使用 SWT.NONE 样式
        ToolBar toolBar = new ToolBar (tool, SWT.NONE);

        //创建保存工具按钮
        ToolItem saveItem = new ToolItem ( toolBar , SWT.PUSH);
        //设置工具按钮的图片
        saveItem.setImage( ImageFactory.loadImage( display , ImageFactory.SAVE_EDIT));
        //设置工具按钮上的文字
        saveItem.setText("保存");
        //设置工具按钮上的提示信息
        saveItem.setToolTipText("保存");
        //创建打印工具按钮
        ToolItem printItem = new ToolItem ( toolBar , SWT.PUSH);
        printItem.setImage( ImageFactory.loadImage( display , ImageFactory.PRINT_EDIT));
        printItem.setText("打印");
        printItem.setToolTipText("打印");
        //创建帮助工具按钮
        ToolItem helpItem = new ToolItem ( toolBar , SWT.PUSH);
        helpItem.setImage( ImageFactory.loadImage( display , ImageFactory.HELP_CONTENTS));
        helpItem.setText("帮助");
        helpItem.setToolTipText("帮助");

        toolBar.pack ();
        Text content = new Text (shell,SWT.MULTI);
        content.setLayoutData( new GridData(SWT.FILL,SWT.FILL,true,true));

        shell.setSize( new Point ( 200,150 ));
        shell.open ();
        while (!shell.isDisposed()) {
            if (!display.readAndDispatch ()) display.sleep ();
        }
        //释放图片资源
    }
}
```

```
ImageFactory.dispose();  
display.dispose ();  
}  
}
```

使用工具栏控件时应该注意以下几个问题：

- ❑ 一般情况下，设计工具栏时有两种风格：一种是图标加文字，也就是图 9.8 显示的风格，这样的工具栏很直观，缺点是占用的空间太大；另一种是只显示图标，虽然节省了空间，但这又会使用户不能很快理解工具栏按钮的用途。所以一个最好的解决方案就是结合提示（ToolTip）来使用，通过 `setToolTipText(String string)` 方法可以设置提示。这样，当用户将鼠标移动到工具栏的按钮上时，会出现提示文字来提示用户该按钮的用途。
- ❑ 使用工具栏避免不了使用大量的图片，随着图片的增多又会占用大量的资源，所以图片很多时，需要一种很好的管理机制来管理这些图片，当图片不用时需要释放掉。有关图片资源的管理将在 9.3.1 节介绍。

9.3.1 工具栏图片资源的管理

在前面学习的使用图片的方法一般都是当需要图片时就创建一个图片对象，然后使用完图片后释放。例如以下代码：

```
Image image = new Image(display,"F:\\tools\\samples.gif");  
//使用图片资源后释放，中间的程序省略  
image.dispose();
```

如果只有一两个图片，这种直接创建图片对象然后释放的办法是可行的。但当图片增多时也要一个一个地创建和释放吗？可想而知，代码将变得很复杂。所以有必要将图片的管理集中到一起，进行统一的管理。

在上边的工具栏程序中，已经使用了图片管理方案，即使用 `ImageFactory` 类作为管理图片的类。以下是该类的代码：

ImageFactory.java

```
package com.fengmanfei.util;  
  
import java.util.Enumeration;  
import java.util.Hashtable;  
import org.eclipse.swt.graphics.Image;  
import org.eclipse.swt.widgets.Display;  
  
public class ImageFactory {  
  
    //将构造方法设置为 private，禁止创建该类的实例  
    private ImageFactory() {}  
    //图片保存的绝对地址
```

```

    public static final String REAL_PATH = "E:\\Documents and Settings\\jan\\workspace\\
    SWTWork\\icons\\full\\etool16\\";
    //一些图片名称的常量
    public static final String DELETE_EDIT = "delete_edit.gif";
    public static final String SAVE_EDIT = "save_edit.gif";
    public static final String SCOPY_EDIT = "copy_edit.gif";
    public static final String CUT_EDIT = "cut_edit.gif";
    public static final String PRINT_EDIT = "print_edit.gif";
    public static final String HELP_CONTENTS = "help_contents.gif";
    //使用 Hashtable 保存使用的图片资源
    private static Hashtable htlImage = new Hashtable();
    //加载图片.首先从 htlImage 中获得图片对象
    //如果没有, 则加载新的图片并放入到 htlImage
    public static Image loadImage(Display display, String imageName) {
        Image image = (Image) htlImage.get(imageName.toUpperCase());
        if (image == null) {
            image = new Image(display, REAL_PATH + imageName);
            htlImage.put(imageName.toUpperCase(), image);
        }
        return image;
    }

    //释放 htlImage 中的所有的图片资源
    public static void dispose() {
        Enumeration e = htlImage.elements();
        while (e.hasMoreElements()) {
            Image image = (Image) e.nextElement();
            if (!image.isDisposed())
                image.dispose();
        }
    }
}

```

该类的实现需要说明的有以下几点:

- ❑ 将所有图片资源使用一个 Hashtable 保存, 当使用图片时, 首先要在 Hashtable 中查找是否已经加载了该图片。如果已经加载了, 那么就直接使用。如果还未加载, 则先加载图片, 并放入到 Hashtable 中, 以便以后使用。
- ❑ 释放资源时, 只需调用 dispose()方法将 Hashtable 保存的图片资源释放就可以了。
- ❑ 有关图片的路径问题, 该类定义了一些图片的常量, 方便外部调用。

使用该类时, 创建图片和释放图片资源的方法的代码如下:

```

//加载一个图片, 使用 ImageFactory.SAVE_EDIT 的常量
Image image1 = ImageFactory.loadImage( display , ImageFactory.SAVE_EDIT);
//加载另一个图片, 使用 ImageFactory.PRINT_EDIT 常量
Image image2 = ImageFactory.loadImage( display , ImageFactory.PRINT_EDIT);
//使用完图片后, 最后释放
ImageFactory.dispose();

```

将图片资源集中管理，并且将图片文件名设置为常量，以后要是修改图片就很容易了。

注意：这里使用了 Eclipse 自带的一些图片，Eclipse 的图标保存在 Eclipse 根文件目录 `plugins\org.eclipse.ui_3.1.2.jar` 中。如果使用，可以将其解压缩，复制到其他位置就可以了。当然，解压缩的办法有很多，最简单的就是使用 winRAR 软件直接解压缩。

9.3.2 工具栏的不同样式

1. 垂直工具栏：SWT.VERTICAL

默认情况下，工具栏设置为水平显示的 `SWT.HORIZONTAL` 样式。但也可以选择设置为垂直方向的工具栏，使用常量 `SWT.VERTICAL`。例如，创建一个垂直的工具栏的代码如下：

```
ToolBar toolBar = new ToolBar(tool, SWT.VERTICAL|SWT.NONE);
```

垂直工具栏的效果如图 9.10 所示。

2. 平滑样式：SWT.FLAT

平滑样式的工具栏是最接近我们常见的工具栏样式，鼠标不在按钮上时，呈现出平面的状态，当鼠标移动到按钮上时，即显示出按钮的边框。如图 9.11 所示为平滑样式工具栏的效果图。

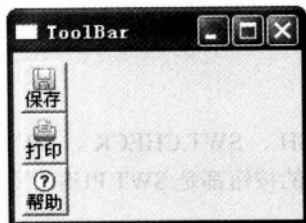


图 9.10 垂直工具栏效果图



图 9.11 平滑效果的工具栏 (SWT.FLAT)

3. 右侧显示文字样式：SWT.RIGHT

默认情况下，如果既有图标又有文字，文字会显示在图标的下方，但当使用 `SWT.RIGHT` 样式后，文字会显示在图标的右侧。例如，创建一个既平滑显示，又右侧显示文字的工具栏，代码如下：

```
ToolBar toolBar = new ToolBar(tool, SWT.RIGHT|SWT.FLAT);
```

右侧显示文字的工具栏效果如图 9.12 所示。

4. 带阴影的工具栏：SWT.SHADOW_OUT

使用 `SWT.SHADOW_OUT` 可以使工具栏产生阴影效果。如图 9.13 所示为显示阴影的工具栏效果图。



图 9.12 右侧显示文字 (SWT.RIGHT)



图 9.13 显示阴影的工具栏 (SWT.SHADOW_OUT)

5. 带边框的工具栏: SWT.BORDER

使用 SWT.BORDER 可以使工具栏产生带边框的效果。如图 9.14 所示为显示边框的工具栏效果图。



图 9.14 带有边框的工具栏 (SWT.BORDER)

9.3.3 工具栏按钮的不同样式

同菜单栏一样，工具栏按钮也有 SWT.PUSH、SWT.CHECK、SWT.RADIO、SWT.SEPARATOR 和 SWT.DROP_DOWN。前面使用的按钮都是 SWT.PUSH 风格的。这里不多介绍，详细看一下其他样式。

1. 多选工具栏按钮和单选工具栏按钮: SWT.CHECK 和 SWT.RADIO

多选工具栏按钮和单选工具栏按钮在外观上没有什么区别，当按钮处于选中状态时呈下压的状态。它们的主要区别是：多选按钮可以同时选中多个，但单选按钮只能在几个按钮之间选中一个。图 9.15 和图 9.16 显示了不同样式工具栏按钮的效果图。



图 9.15 多选工具栏按钮 (SWT.CHECK)



图 9.16 单选工具栏按钮 (SWT.RADIO)

另外，设置和获取按钮是否选中的方法是 `getSelection()` 和 `setSelection(boolean selected)`。

2. 工具栏分割线按钮: SWT.SEPARATOR

在工具栏中创建一个分割线的方法很简单，只需要使用 SWT.SEPARATOR 样式即可。代码如下：


```
new ToolItem ( toolBar , SWT.SEPARATOR);
```

例如，为以上的工具栏添加分割线后的效果如图 9.17 所示。

3. 带有下拉按钮的工具栏按钮：SWT.DROP_DOWN

带有下拉按钮的工具栏的外观如图 9.18 所示。虽然具有带下拉按钮的外观，但单击后并不会有什么反应。要实现一些功能，需要配合程序来使用。例如，下拉菜单通常是配合 9.3.2 节学习的弹出式菜单来使用的，当单击下拉按钮后，弹出一个菜单，如图 9.19 所示。



图 9.17 带分割线的工具栏



图 9.18 带下拉按钮的工具栏



图 9.19 单击下拉按钮弹出菜单

该程序的功能实现需要在上个程序 `ToolBarSample.java` 中稍作修改。这里新建一个文件，命名为 `ToolBarSample1.java`。以下显示的只是修改后的主要代码：

ToolBarSample1.java

```
//前面程序与 ToolBarSample.java 代码类似，在此省略
//创建帮助工具按钮，定义为下拉式。使用 SWT.DROP_DOWN 样式
final ToolItem helpItem = new ToolItem ( toolBar , SWT.DROP_DOWN);
helpItem.setImage( ImageFactory.loadImage( display , ImageFactory.HELP_CONTENTS));
helpItem.setText("帮助");
helpItem.setToolTipText("帮助");

//定义一个弹出式菜单，使用常量 SWT.POP_UP
final Menu helpMenu = new Menu( shell , SWT.POP_UP);
//定义菜单中的菜单项
MenuItem welcomeItem = new MenuItem(helpMenu,SWT.PUSH);
welcomeItem.setText("欢迎");
welcomeItem.setImage( ImageFactory.loadImage( display , ImageFactory.ECLIPSE ));

new MenuItem( helpMenu ,SWT.SEPARATOR);

MenuItem updateItem = new MenuItem(helpMenu,SWT.PUSH);
updateItem.setText("在线更新");
```

```
MenuItem aboutItem = new MenuItem(helpMenu, SWT.PUSH);
aboutItem.setText("关于我们");
aboutItem.setImage( ImageFactory.loadImage( display , ImageFactory.SAMPLES) );
//为下拉工具栏注册单击下拉箭头事件
helpItem.addListener(SWT.Selection, new Listener(){

    public void handleEvent(Event event) {
        //如果触发的事件为单击按钮
        if (event.detail==SWT.ARROW)
        {
            //获得帮助按钮所在坐标位置和大小，相对于父窗口
            Rectangle rect = helpItem.getBounds ();
            //计算菜单出现的起始位置
            Point pt = new Point (rect.x, rect.y + rect.height);
            //将该点转化为与屏幕的相对位置
            pt = toolBar.toDisplay (pt);
            helpMenu.setLocation (pt.x, pt.y); //设置菜单显示的位置
            helpMenu.setVisible (true); //显示菜单
        }
    }
});
toolBar.pack ();
//后边的代码省略
```

实现该程序主要有以下几点需要注意：

- ☐ 弹出菜单的样式要定义为 `SWT.POP_UP`，这样才可以为弹出式。
- ☐ 带有下拉按钮的工具栏按钮的样式要设置为 `SWT.DROP_DOWN`，这样才可以显示带有下拉按钮。
- ☐ 判断是否是单击下拉按钮事件的方法的代码如下：

```
if (event.detail==SWT.ARROW)
```

- ☐ 显示菜单时首先要计算菜单所要出现的位置，然后通过 `setLocation (pt.x, pt.y)` 方法定位菜单，最后使用 `setVisible (true)` 方法将菜单显示出来。

9.3.4 工具栏常用的方法

除了以上程序中使用的方方法外，`ToolBar` 和 `ToolItem` 类还有一些其他常用的方法。

1. `ToolBar` 类的常用方法

- ☐ 根据索引值查找工具栏按钮，返回 `ToolItem` 对象：`getItem(int index)`。
- ☐ 根据所在点的位置查找工具栏按钮，返回 `ToolItem` 对象：`getItem(Point point)`。
- ☐ 获得所有工具栏按钮的个数，返回 `int`：`getItemCount()`。
- ☐ 获得所有工具栏按钮数组，返回 `ToolItem` 数组对象：`getItems()`。

2. ToolItem 类的常用方法

- ❑ 设置鼠标放上后的图片: `setHotImage(Image image)`。通常可以利用这个方法使按钮产生动态效果。
- ❑ 设置按钮不可用状态时的图片: `setDisabledImage(Image image)`。
- ❑ 设置是否可用: `setEnabled(boolean enabled)`, 默认为 `true`。

9.4 可拖动的工具栏 (CoolBar 和 CoolItem)

可拖动的工具栏是最接近经常使用的工具栏, 这种工具栏可以通过拖动放置到所需要的位置。如图 9.20 所示为可拖动的工具栏效果图。

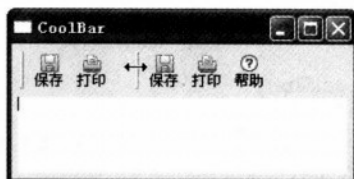


图 9.20 可拖动工具栏效果图

实现这样的可拖动效果的工具栏代码如下:

CoolBarSample.java

```
package com.fengmanfei.ch9;
import org.eclipse.swt.SWT;
import org.eclipse.swt.graphics.Point;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.widgets.*;

import com.fengmanfei.util.ImageFactory;

public class CoolBarSample {

    public static void main(String[] args) {
        Display display = new Display ();
        Shell shell = new Shell (display);
        shell.setText("CoolBar");
        shell.setLayout( new GridLayout());
        Composite tool = new Composite( shell ,SWT.NONE);
        tool.setLayoutData(new GridData(SWT.LEFT,SWT.TOP,true ,false));
        //定义可拖动的工具栏对象
        CoolBar coolBar = new CoolBar (tool, SWT.FLAT);
        //创建两个工具栏项
        createItem(display ,coolBar);
        createItem(display ,coolBar);
        coolBar.pack ();
    }
}
```

```

Text content = new Text (shell,SWT.MULTI);
content.setLayoutData( new GridData(SWT.FILL,SWT.FILL,true,true));

shell.setSize( new Point ( 200,150 ));
shell.open ();
while (!shell.isDisposed()) {
    if (!display.readAndDispatch ()) display.sleep ();
}
//释放图片资源
ImageFactory.dispose();
display.dispose ();
}

public static CoolItem createItem(Display display ,CoolBar coolBar) {
    //创建一个工具栏
    ToolBar toolBar = createToolBar(display, coolBar);
    //获得工具栏的大小
    Point size = toolBar.getSize();
    //定义一个 CoolItem 对象
    CoolItem item = new CoolItem(coolBar, SWT.NONE);
    //将工具栏绑定到这个工具栏项上
    item.setControl(toolBar);
    //计算工具栏项合适的大小
    Point preferred = item.computeSize(size.x, size.y);
    //设置大小
    item.setPreferredSize(preferred);
    return item;
}

public static ToolBar createToolBar(Display display, CoolBar coolBar) {
    ToolBar toolBar = new ToolBar(coolBar, SWT.FLAT);
    //创建保存工具按钮
    ToolItem saveItem = new ToolItem ( toolBar , SWT.PUSH);
    //设置工具按钮的图片
    saveItem.setImage( ImageFactory.loadImage( display , ImageFactory.SAVE_EDIT));
    //设置工具按钮上的文字
    saveItem.setText("保存");
    //设置工具按钮上的提示信息
    saveItem.setToolTipText("保存");
    //创建打印工具按钮
    ToolItem printItem = new ToolItem ( toolBar , SWT.PUSH);
    printItem.setImage( ImageFactory.loadImage( display , ImageFactory.PRINT_EDIT));
    printItem.setText("打印");
    printItem.setToolTipText("打印");
    //创建帮助工具按钮
    ToolItem helpItem = new ToolItem ( toolBar , SWT.PUSH);
    helpItem.setImage( ImageFactory.loadImage( display , ImageFactory.HELP_CONTENTS));
    helpItem.setText("帮助");
    helpItem.setToolTipText("帮助");
    toolBar.pack();
}

```

```

        return toolBar;
    }
}

```

实现该程序需要注意以下几点：

- ❑ CoolBar 与 CoolItem 是一对多的关系。CoolItem 与 ToolItem 不同，CoolItem 是通过方法 `setControl(Control control)` 将控件与某个控件绑定的。例如这个程序中，是将一个工具栏 (ToolBar) 绑定到 CoolItem 上的，当然也可以将其他控件，如按钮 (Button) 等控件绑定。
- ❑ 并不是说将控件绑定到 CoolItem 上就可以显示控件了，而需要计算控件合适的大小，重新设置 CoolItem 的大小，详细代码请参考代码注释部分。
- ❑ 将创建菜单的功能写成一个方法，这样有助于代码重用。这种创建控件的方法读者应该熟悉，因为在实际项目的过程中都是使用这种方法来创建的。

9.4.1 带有下拉选项的工具栏

CoolItem 同 ToolItem 一样，可选的风格也有 `SWT.DROP_DOWN`。例如，以下代码可以创建一个带有下拉按钮的 CoolItem。

```
CoolItem item = new CoolItem(coolBar, SWT.DROP_DOWN);
```

如图 9.21 所示，显示带有下拉按钮的 CoolItem。



图 9.21 带有下拉按钮的工具栏

当然，与 ToolItem 一样，可以配合菜单来使用。这里不再详细介绍，请读者参考 ToolItem 的示例程序。

9.4.2 常用的方法

除了与 ToolBar 类似的方法外，以下还列举了 CoolBar 和 CoolItem 的一些常用的方法。

1. CoolBar 的常用方法

- ❑ 返回所有 CoolItem 的排列顺序，返回类型为 int 数组：`getItemOrder()`。
- ❑ 设置工具栏是否可以拖动：`setLocked(boolean locked)`。如果设置为 true，表示处于锁定状态，工具栏不能拖动。

2. CoolItem 的常用方法

- ❑ 设置最小的大小：`setMinimumSize(int width, int height)`和 `setMinimumSize(Point size)`。
- ❑ 设置最佳的大小：`setPreferredSize(int width, int height)`和 `setPreferredSize(Point size)`。

9.5 系统托盘 (Tray 和 TrayItem)

系统托盘是放置在右下角的图标。如图 9.22 所示为 Windows XP 系统托盘的样式。通常会将应用程序隐藏在系统托盘上。例如聊天工具 MSN，通常会在系统托盘上显示当前的状态，并且最小化时也会隐藏到系统托盘上。

SWT 中使用 Tray 和 TrayItem 这两个类来创建系统托盘和系统托盘项。下面以一个例子来介绍 Tray 和 TrayItem 的具体使用。如图 9.23 所示为该程序的最终效果图。创建一个系统托盘，右击系统托盘项时显示一个菜单。



图 9.22 Windows XP 系统托盘

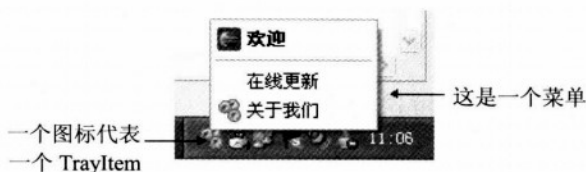


图 9.23 系统托盘程序效果图

以下为该程序的最终实现代码：

TraySample.java

```
package com.fengmanfei.ch9;
import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Event;
import org.eclipse.swt.widgets.Listener;
import org.eclipse.swt.widgets.Menu;
import org.eclipse.swt.widgets.MenuItem;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.widgets.Tray;
import org.eclipse.swt.widgets.TrayItem;

import com.fengmanfei.util.ImageFactory;

public class TraySample {

    public static void main(String[] args) {
        Display display = new Display();
        final Shell shell = new Shell(display);
        // 创建系统托盘
        final Tray tray = display.getSystemTray();
        // 如果系统不支持托盘部件
        if (tray == null) {
            System.out.println("该系统不支持系统托盘");
            return;
        }
    }
}
```



```

    }
    final Menu trayMenu = createTrayMenu(display, shell);
    // 创建系统托盘的工作项
    final TrayItem item = new TrayItem(tray, SWT.NONE);
    item.setToolTipText("这是一个 TrayItem");
    // 设置显示系统托盘项的图标, 显示在桌面的右下角
    item.setImage(ImageFactory.loadImage(display, ImageFactory.SAMPLES));
    // 集中处理事件
    Listener listener = new Listener(){

        public void handleEvent(Event event) {

            if ( event.type==SWT.Show )//当显示系统托盘时
                System.out.println("显示");
            else if ( event.type==SWT.Hide )//当隐藏系统托盘时
                System.out.println("隐藏");
            else if ( event.type==SWT.Selection )//当单击系统托盘时
                System.out.println("选中");
            else if ( event.type==SWT.DefaultSelection )//当双击系统托盘时
                System.out.println("默认选中");
            else if ( event.type==SWT.MenuDetect )//当右击系统托盘时
                trayMenu.setVisible(true);//设置菜单为显示状态

        }

    };
    // 为该系统托盘项注册事件
    item.addListener( SWT.Show, listener );
    item.addListener( SWT.Hide, listener );
    item.addListener( SWT.Selection, listener );
    item.addListener( SWT.DefaultSelection, listener );
    item.addListener( SWT.MenuDetect, listener );

    shell.pack();
    shell.open();
    while (!shell.isDisposed()) {
        if (!display.readAndDispatch())
            display.sleep();
    }
    ImageFactory.dispose();
    display.dispose();
}

private static Menu createTrayMenu(Display display, final Shell shell){
    // 定义一个弹出式菜单, 使用常量 SWT.POP_UP
    final Menu trayMenu = new Menu(shell, SWT.POP_UP);
    // 定义菜单中的菜单项
    MenuItem welcomeItem = new MenuItem(trayMenu, SWT.PUSH);
    welcomeItem.setText("欢迎");
    welcomeItem.setImage(ImageFactory.loadImage(display,

```

```

        ImageFactory.ECLIPSE));

        new MenuItem(trayMenu, SWT.SEPARATOR);

        MenuItem updateItem = new MenuItem(trayMenu, SWT.PUSH);
        updateItem.setText("在线更新");

        MenuItem aboutItem = new MenuItem(trayMenu, SWT.PUSH);
        aboutItem.setText("关于我们");
        aboutItem.setImage(ImageFactory.loadImage(display, ImageFactory.SAMPLES));

        trayMenu.setDefaultItem(welcomeItem);
        return trayMenu;
    }
}

```

综合以上程序，总结一下在使用 Tray 和 TrayItem 时应该注意的问题：

- ❑ 创建系统托盘时要通过 Display 对象的 `getSystemTray()` 方法的操作系统不同，对系统托盘的支持也不同。若系统不支持系统托盘，将返回 `null`。
- ❑ Tray 和 TrayItem 也是一对多的关系，同 Menu 和 MenuItem 的关系一样。
- ❑ TrayItem 通常与一个弹出式菜单连用，配合 TrayItem 的事件，将菜单显示出来。

TrayItem 的一些事件说明，请读者仔细区分不同的事件。

- ❑ `SWT.Show` 和 `SWT.Hide`：当系统托盘项增多时，可能会隐藏一些图标，如图 9.24 所示为隐藏图标的效果图。

`SWT.Show` 和 `SWT.Hide` 就是当单击按钮时所触发的事件。


 **注意：**在 Windows XP 操作系统下，要保证任务栏属性中选中“隐藏不活动图标”，才可以有显示和隐藏的事件发生。查看任务栏属性的方法是：在任务栏上右击，在弹出的快捷菜单中选择“属性”命令，如图 9.25 所示为任务栏属性的设置。



图 9.24 隐藏系统托盘后的效果

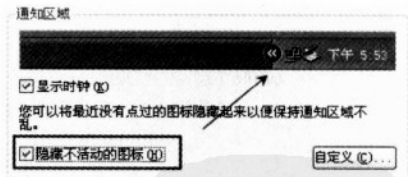


图 9.25 任务栏属性设置

- ❑ `SWT.Selection`：当选定该图标时，也就是单击该图标所触发的事件。
- ❑ `SWT.DefaultSelection`：区别于 `SWT.Selection`，为双击图标时所触发的事件。例如 MSN，双击时会显示主窗口。
- ❑ `SWT.MenuDetect`：也就是在图标上右击时的事件，通常是弹出一个菜单。

9.6 滑动组件

滑动组件一般用于对音量的调节，或对色彩的调节等。SWT 控件中涉及滑动组件的类有滑块（Slider）、刻度条（Scale）和微调按钮（Spinner）等。下面详细介绍这几个控件的使用。

9.6.1 滑块（Slider）

滑块有点像滚动条，带有两个调节的箭头和一个小滑块，通过滑块和箭头设定值的大小。如图 9.26 所示为水平显示的滑块效果图。

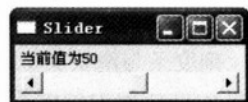


图 9.26 水平滑块

创建一个这样的滑块的代码如下：

SliderSample.java

```
//创建一个标签显示当前所选中的值
final Label sliderValue = new Label ( shell , SWT.NONE);
//创建滑块对象
final Slider slider = new Slider (shell, SWT.HORIZONTAL);
slider.setMaximum(100);//设置最大值为 100
slider.setMinimum(0);//设置最小值为 0
slider.setIncrement(1);//设置每次单击箭头时增长的值
slider.setThumb(10);//设置滑块的大小
slider.setPageIncrement(20);//设置单击滑块空白处一次移动的值
slider.setSelection(50);//设置选中的值
sliderValue.setText( "当前值为"+slider.getSelection());
//为滑块注册事件
slider.addListener (SWT.Selection, new Listener () {
    public void handleEvent (Event event) {
        String string = "SWT.NONE";
        switch (event.detail) {
            case SWT.DRAG: string = "SWT.DRAG"; break;
            case SWT.HOME: string = "SWT.HOME"; break;
            case SWT.END: string = "SWT.END"; break;
            case SWT.ARROW_DOWN: string = "SWT.ARROW_DOWN"; break;
            case SWT.ARROW_UP: string = "SWT.ARROW_UP"; break;
            case SWT.PAGE_DOWN: string = "SWT.PAGE_DOWN"; break;
            case SWT.PAGE_UP: string = "SWT.PAGE_UP"; break;
        }
        System.out.println ("触发的事件为 " + string);
        sliderValue.setText( "当前值为"+slider.getSelection());
    }
});
```

使用 Slider 对象时需要注意以下几个问题:

- ❑ 对 Slider 属性的一些设置, 例如最大值和最小值及每次单击按钮时所增长的值等, 请读者参阅程序的代码注释部分。
- ❑ Slider 所涉及的一些事件, 请读者参考程序的实现部分。
- ❑ 还可以使用 SWT.VERTICAL 样式常量创建一个垂直显示的滑块。

9.6.2 刻度条 (Scale)

刻度条与滑块类似, 各种属性差不多, 只不过是外观不同。

如图 9.27 所示为刻度条的最终效果图。

创建这样的一个刻度条的代码如下:

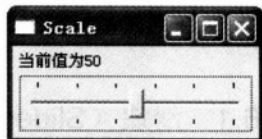


图 9.27 刻度条效果图

ScaleSample.java

```
//创建刻度条对象
final Scale scale = new Scale(shell, SWT.HORIZONTAL);
scale.setMaximum(100); //设置最大值为 100
scale.setMinimum(0); //设置最小值为 0
scale.setIncrement(1); //设置拖动滑块时增长的值
scale.setPageIncrement(20); //设置单击滑块空白处一次移动的值
scale.setSelection(50); //设置当前选中的值
sliderValue.setText( "当前值为"+scale.getSelection());
scale.addListener (SWT.Selection, new Listener () {
    public void handleEvent (Event event) {
        sliderValue.setText( "当前值为"+scale.getSelection());
    }
});
```

9.6.3 微调按钮 (Spinner)

微调按钮基本原理与滑块和刻度条类似, 但是可以显示带小数位数的值, 适合调节精度高的值来使用。如图 9.28 所示为一个微调按钮。

创建这样的一个微调按钮的代码如下:

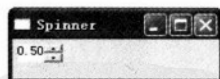


图 9.28 微调按钮

SpinnerSample.java

```
//创建微调按钮对象
final Spinner spinner = new Spinner(shell, SWT.NONE);
spinner.setMaximum(100); //设置最大值为 100
spinner.setMinimum(0); //设置最小值为 0
spinner.setIncrement(1); //设置每次单击箭头时增长的值
spinner.setPageIncrement(20);
```

```
spinner.setSelection(50); //设置当前选中的值
spinner.setDigits(2); //设置小数点两位
```

使用微调按钮类时应注意以下几点:

- ❑ 微调按钮可以通过 `setDigits(int value)` 方法来设定显示的小数点的位数, 默认值为 0。
- ❑ 微调按钮还有其他样式, 如只读样式 `SWT.READ_ONLY` 和带边框样式 `SWT.BORDER`。如图 9.29 和图 9.30 所示为两种样式的示意图。



图 9.29 只读样式 `SWT.READ_ONLY`

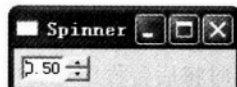


图 9.30 边框样式 `SWT.BORDER`

9.7 进度条 (ProgressBar)

进度条 (ProgressBar) 也是很常见的控件, 一般在运行较大的程序时会用一个进度条, 用来告诉用户后台程序运行的进度情况。如图 9.31 所示即为一个进度条示意图。



图 9.31 进度条示意图

创建这样的一个动态进度条的代码如下:

ProgressBarSample.java

```
//创建滚动条实例
final ProgressBar progressBar = new ProgressBar(shell, SWT. HORIZONTAL);
progressBar.setMaximum(100); //设置最大值
progressBar.setMinimum(0); //设置最小值
final int maximum = progressBar.getMaximum(); //获取最大值
final int minimus = progressBar.getMinimum(); //获取最小值
//创建一个线程, 该线程每 0.1 秒更新一次滚动条的值
Runnable runnable = new Runnable() {
    public void run() {
        for (int i = minimus; i < maximum; i++) {
            try {
                //让线程睡眠 0.1 秒
                Thread.sleep(100);
            } catch (InterruptedException e) {
            }
            //让 UI 线程更新滚动条的值
            display.asyncExec(new Runnable() {
                public void run() {
                    if (progressBar.isDisposed())
                        return;
                }
            });
        }
    }
};
```



```

        progressBar.setSelection(progressBar.getSelection() + 1);
    }
    });
}
};
//启动这个线程
new Thread(runnable).start();

```

就进度条本身来说，它的使用方法并不难。但进度条通常要与线程同时使用，即后台运行一个线程，同时将后台程序运行的状态以进度条的方式显示出来。线程的处理是 SWT 程序中一个很重要的知识点，笔者将以一个章节来讲述有关线程的问题。如果读者在这里对线程的理解有些困难，也不用担心，这里只要知道进度条的使用方法就可以了。

进度条除了如图 9.31 所示的 SWT.HORIZONTAL 样式的进度条外，还有一些其他样式的进度条。

- ☐ 平滑型：SWT.SMOOTH，如图 9.32 所示。
- ☐ 垂直型：SWT.VERTICAL，如图 9.33 所示。



图 9.32 平滑型 SWT.SMOOTH

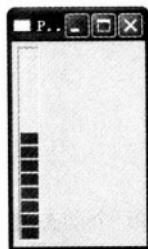


图 9.33 垂直型 SWT.VERTICAL

9.8 对话框

SWT 中的对话框有消息提示框（MessageBox）、文件目录对话框（DirectoryDialog）、文件对话框（FileDialog）、颜色对话框（ColorDialog）、字体对话框（FontDialog）和打印对话框（PrintDialog）等。所有的这些对话框都是由 org.eclipse.swt.widgets.Dialog 继承而来的。对话框的类的继承关系图如图 9.34 所示。

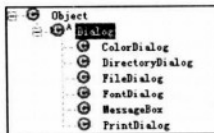


图 9.34 Dialog 对话框类继承关系图

当然，JFace 中也提供了功能更强大的对话框。这里不多作介绍，在 JFace 篇将会详细讲述。

9.8.1 消息提示框 (MessageBox)

消息提示框是很常用的一种对话框，当删除文件时一般会弹出一个确认对话框来询问是否删除。如图 9.35 所示即为一个警告式的对话框。

创建这样的一个消息提示框的代码如下：

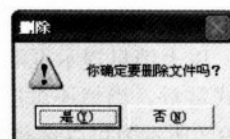


图 9.35 删除对话框

DialogSample.java

```
Button b1 = new Button ( shell,SWT.NONE);
b1.setText("消息提示框");
b1.addSelectionListener( new SelectionAdapter(){
    public void widgetSelected(SelectionEvent e) {
        //创建消息框对象，使用警告图标并显示"是"和"否"按钮
        MessageBox box = new MessageBox( shell ,SWT.ICON_ WARNING|SWT. YES|
SWT.NO);
        //设置对话框的标题
        box.setText("删除");
        //设置对话框显示的消息
        box.setMessage("您确定要删除文件吗? ");
        //打开对话框，将返回值赋给 choice
        int choice = box.open();
        //打印出所选择的值
        if (choice==SWT.YES)
            System.out.print("Yes");
        else if ( choice==SWT.NO)
            System.out.print("No");
    }
});
```

使用对话框类时应注意以下几点。

1. 图标的显示

在上面的例子中显示的是警告图标，使用的是 SWT.ICON_WARNING 样式常量。还可以使用其他图标，可用的样式常量如表 9.1 所示。

表 9.1 图标常量

图 标 常 量	说 明	效 果 图
SWT.ICON_ERROR	错误提示的图标	
SWT.ICON_INFORMATION	信息提示的图标	
SWT.ICON_QUESTION	询问提示的图标	
SWT.ICON_WARNING	警告的图标	

2. 提示框中的按钮

以上消息提示框的例子中显示了“是”和“否”两个按钮,使用的是 SWT.YES 和 SWT.NO 样式常量。当然还可以根据需求显示不同的按钮,只要使用不同按钮的样式常量即可。如表 9.2 所示为可以选择组合的按钮样式。

表 9.2 按钮的组合样式

按 钮 组 合	说 明
SWT.OK	只显示“确定”按钮
SWT.OK SWT.CANCEL	显示“确定”和“取消”按钮
SWT.YES SWT.NO	显示“是”和“否”按钮
SWT.YES SWT.NO SWT.CANCEL	显示“是”、“否”和“取消”按钮
SWT.RETRY SWT.CANCEL	显示“重试”和“取消”按钮
SWT.ABORT SWT.RETRY SWT.IGNORE	显示“终止”、“重试”和“忽略”按钮

3. 打开消息提示框

MessageBox 类常用的方法请读者查看代码的注释部分,在设置好属性后要调用 open() 方法才可以将窗口打开,并返回一个 int 型数值。判断用户所选择的按钮,要判断该返回值。

9.8.2 文件目录对话框 (DirectoryDialog)

文件目录对话框一般出现在保存文件时,让用户选择一个保存的文件目录。如图 9.36 所示为一个打开的文件目录对话框。



图 9.36 文件目录对话框

创建这样的一个对话框的代码如下：

DialogSample.java

```
Button b2 = new Button ( shell,SWT.NONE);
b2.setText("目录选取对话框");
b2.addSelectionListener( new SelectionAdapter(){
    public void widgetSelected(SelectionEvent e) {
        //创建文件目录对话框对象
```

```

DirectoryDialog dialog = new DirectoryDialog(shell);
//设置显示在窗口上方的提示信息
dialog.setMessage("请选择所要保存的文件夹");
//设置对话框的标题
dialog.setText("选择文件目录");
//设置打开时默认的文件目录
dialog.setFilterPath("C:\\");
//打开窗口，返回用户所选的文件目录
String saveFile = dialog.open();
if ( saveFile != null )
{
    //创建一个 File 对象
    File directory = new File(saveFile);
    System.out.print(directory.getPath());
}
}
};

```

在使用文件目录对话框时应注意以下几个问题：

- ❑ 可通过 `setFilterPath(String string)` 方法来设置打开对话框时默认的文件目录。
- ❑ 使用 `open()` 方法打开对话框后，返回文件目录的路径字符串。

9.8.3 文件对话框（FileDialog）

文件对话框一般出现在打开或保存一个文件时。如图 9.37 所示为一个打开样式的对话框。



图 9.37 打开文件对话框

创建这样的文件对话框的代码如下：

DialogSample.java

```

Button b3 = new Button ( shell,SWT.NONE);
b3.setText("文件对话框");

```

```

b3.addSelectionListener( new SelectionAdapter(){
    public void widgetSelected(SelectionEvent e) {
        //创建一个打开对话框，样式设置为 SWT.OPEN
        FileDialog dialog = new FileDialog(shell,SWT.OPEN);
        //设置打开默认的路径
        dialog.setFilterPath(System.getProperty("java.home"));
        //设置所打开文件的扩展名
        dialog.setFilterExtensions(new String[] {"*.txt", "*.xml"});
        //设置显示到下拉框中的扩展名的名称
        dialog.setFilterNames( new String[]{"Text Files (*.txt)", "All Files (*.*)"});
        //打开窗口，返回用户所选的文件目录
        String file = dialog.open();
        if ( file != null )
        {
            System.out.print(file);
        }
    }
});

```

使用文件对话框时应该注意以下几个问题。

1. 文件对话框所使用的样式

- ❑ SWT.OPEN: 显示带有“打开”按钮的文件对话框，如图 9.38 所示。

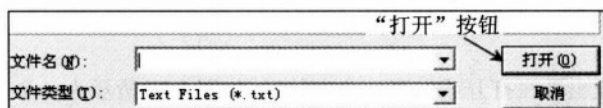


图 9.38 打开文件对话框

- ❑ SWT.SAVE: 显示带有“保存”按钮的文件对话框，如图 9.39 所示。

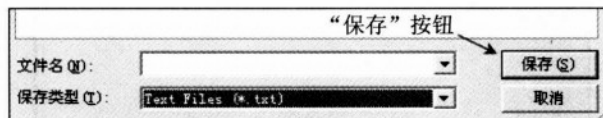


图 9.39 保存文件对话框

- ❑ SWT.MULTI: 可以同时选中多个文件的文件对话框，如图 9.40 所示。

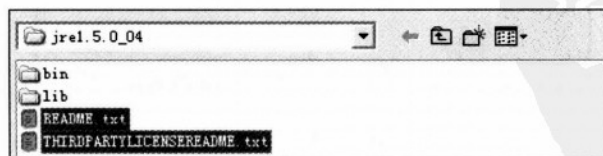


图 9.40 同时选中多个文件

2. 文件对话框常用的方法

- ❑ 设置打开窗口默认选中的文件名: `setFileName(String string)`。

- ❑ 设置打开窗口默认的目录名: `setFilterPath(String string)`。
- ❑ 设置可选文件的扩展名: `setFilterExtensions(String[] extensions)`。
- ❑ 设置可选文件扩展名的显示名称: `setFilterNames(String[] names)`与 `setFilterExtensions(String[] extensions)`方法对应。

9.8.4 颜色对话框 (ColorDialog)

在选用颜色时,通常会使用颜色对话框,显示自定义颜色的颜色对话框,如图 9.41 所示。

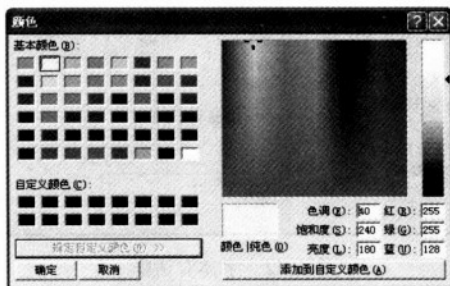


图 9.41 颜色对话框

创建该颜色对话框的代码如下:

DialogSample.java

```
Button b4 = new Button ( shell,SWT.NONE);
b4.setText("颜色对话框");
b4.addSelectionListener( new SelectionAdapter(){
    public void widgetSelected(SelectionEvent e) {
        //创建一个颜色对话框
        ColorDialog dialog = new ColorDialog(shell);
        //设置默认选中的颜色
        dialog.setRGB( new RGB( 255 ,255 ,128));
        //打开对话框, 将选中的颜色返回给 RGB 对象
        RGB rgb = dialog.open();
        if ( rgb != null )
        {
            System.out.print(rgb);
            //创建颜色对象
            Color color = new Color( display , rgb );
            //在使用完颜色对象后, 释放资源
            color.dispose();
        }
    }
});
```

9.8.5 字体对话框 (FontDialog)

同颜色对话框一样，可以使用系统自带的字体对话框。如图 9.42 所示为字体对话框的示意图。它的用法与颜色对话框也差不多。

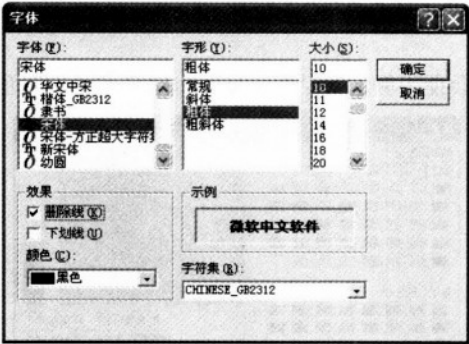


图 9.42 字体对话框

创建该字体对话框的代码如下：

DialogSample.java

```
Button b5 = new Button ( shell,SWT.NONE);
b5.setText("字体对话框");
b5.addSelectionListener( new SelectionAdapter(){
    public void widgetSelected(SelectionEvent e) {
        //创建一个字体对话框
        FontDialog dialog = new FontDialog (shell);
        //设置默认选中的字体
        dialog.setRGB( new RGB( 255 ,255 ,128));
        //打开对话框，将选中的字体返回给 fontData 对象
        FontData fontData = dialog.open();
        if ( fontData != null )
        {
            System.out.print(fontData);
            //创建字体对象
            Font font = new Font( display , fontData );
            //在使用完字体对象后，释放资源
            font.dispose();
        }
    }
});
```

使用字体对话框时，还有以下几个常用的方法：

- ❑ 获得字体对话框的所有字体，返回 FontData 数组：getFontList()。
- ❑ 获得当前选中字体的颜色，返回 RGB 对象：getRGB()。

9.8.6 打印对话框 (PrintDialog)

打印对话框也是系统对话框。如图 9.43 所示为一个系统的打印对话框。



图 9.43 打印对话框

创建这样的一个打印对话框的代码如下：

DialogSample.java

```
Button b6 = new Button ( shell,SWT.NONE);
b6.setText("打印对话框");
b6.addSelectionListener( new SelectionAdapter(){
    public void widgetSelected(SelectionEvent e) {
        //创建一个打印对话框
        PrintDialog dialog = new PrintDialog (shell);
        //打开对话框，将选中的字体返回给 PrinterData 对象
        PrinterData printData = dialog.open();
        if ( printData != null )
        {
            //创建打印对象
            Printer printer = new Printer( printData );
            //在使用打印对象后，释放资源
            printer.dispose();
        }
    }
});
```

使用打印对话框时，还有以下几种常用的方法：

- ❑ 获得起止页数：getStartPage()和 getEndPage()。
- ❑ 获得页面范围：getScope()。返回值有 SWT.ALL_PAGES 表示全部、SWT.PAGE_RANGE 表示起止页、SWT.SELECTION 表示选择当前页。
- ❑ 获得是否选择“打印到文件”选项：getPrintToFile()。

9.9 表格 (Table、TableItem 和 TableColumn)

表格是最常用的控件，能够将大量的数据以表格的形式显示出来，如图 9.44 所示为一个普通的表格。SWT 中的表格是用 Table、TableItem 和 TableColumn 对象来显示表格。



图 9.44 表格 (Table) 示意图

创建这样一个表格的代码如下：

TableSample.java

```
package com.fengmanfei.ch9;

import org.eclipse.swt.SWT;
import org.eclipse.swt.custom.ViewForm;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.widgets.*;
import com.fengmanfei.util.ImageFactory;

public class TableSample {

    private Shell sShell = null;
    private ViewForm viewForm = null;
    private ToolBar toolBar = null;
    private Composite composite = null;
    private Table table = null;

    //创建 ViewForm 面板放置工具栏和表格
    private void createViewForm() {
        viewForm = new ViewForm(sShell, SWT.NONE);
        viewForm.setTopCenterSeparate(true);
        createToolBar();
        viewForm.setTopLeft(toolBar);
        createComposite();
        viewForm.setContent(composite);
    }

    //创建工具栏
    private void createToolBar() {
```

```

    toolBar = new ToolBar(viewForm, SWT.FLAT);
    final ToolItem add = new ToolItem(toolBar, SWT.PUSH);
    add.setText("添加");
    add.setImage( ImageFactory.loadImage( toolBar.getDisplay(), ImageFactory.ADD_OBJ));
    final ToolItem del = new ToolItem(toolBar, SWT.PUSH);
    del.setText("删除");
    del.setImage( ImageFactory.loadImage( toolBar.getDisplay(), ImageFactory.DELETE_
OBJ));
    final ToolItem back = new ToolItem(toolBar, SWT.PUSH);
    back.setText("上移");
    back.setImage( ImageFactory.loadImage( toolBar.getDisplay(), ImageFactory.
BACKWARD_NAV));
    final ToolItem forward = new ToolItem(toolBar, SWT.PUSH);
    forward.setText("下移");
    forward.setImage( ImageFactory.loadImage( toolBar.getDisplay(), ImageFactory.
FORWARD_NAV));
    final ToolItem save = new ToolItem(toolBar, SWT.PUSH);
    save.setText("保存");
    save.setImage( ImageFactory.loadImage( toolBar.getDisplay(), ImageFactory.
SAVE_EDIT));
    //工具栏按钮事件处理
    Listener listener = new Listener(){
        public void handleEvent(Event event) {
            //如果单击“添加”按钮，添加一行，在实际的项目实现中通常是接收输入的参
数，然后添加
            //这里为了简单起见，添加固定的一条记录
            if ( event.widget == add )
            {
                TableItem item = new TableItem(table, SWT.NONE);
                item.setText(new String[] { "郑六", "女", "568", "zhengliu@sina.com" });
            }
            //如果单击“删除”按钮
            else if ( event.widget == del)
            {
                //获得当前选中行索引号
                int selectedRow = table.getSelectionIndex();
                //如果选中了某一行，则删除这一行
                if ( selectedRow > -1 )
                    table.remove(selectedRow);
            }
            //如果单击“上移”按钮
            else if ( event.widget == back)
            {
                int selectedRow = table.getSelectionIndex();
                if ( selectedRow > 0 )
                    table.setSelection( selectedRow-1 );//设置选中的行数
            }
            //如果单击“下移”按钮
            else if ( event.widget == forward)

```



```

        {
            int selectedRow = table.getSelectionIndex();
            if ( selectedRow > -1 && selectedRow < table.getItemCount()-1 )
                table.setSelection( selectedRow+1 );//设置选中的行数
        }
        //如果单击“保存”按钮
        else if ( event.widget == save )
        {
            TableItem[] items = table.getItems();
            //保存到文件或数据库中，数据持久化，这里省略
        }
    }

};
//为工具栏的按钮注册事件
add.addListener( SWT.Selection, listener );
del.addListener( SWT.Selection, listener );
back.addListener( SWT.Selection, listener );
forward.addListener( SWT.Selection, listener );
save.addListener( SWT.Selection, listener );
}
//创建放置表格的面板
private void createComposite() {
    GridLayout gridLayout = new GridLayout();
    gridLayout.numColumns = 1;
    composite = new Composite(viewForm, SWT.NONE);
    composite.setLayout(gridLayout);
    createTable();
}
//创建表格
private void createTable() {
    //表格布局
    GridData gridData = new org.eclipse.swt.layout.GridData();
    gridData.horizontalAlignment = SWT.FILL;
    gridData.grabExcessHorizontalSpace = true;
    gridData.grabExcessVerticalSpace = true;
    gridData.verticalAlignment = SWT.FILL;
    //创建表格，使用 SWT.FULL_SELECTION 样式，可同时选中一行
    table = new Table(composite, SWT.FULL_SELECTION);
    table.setHeaderVisible(true);//设置显示表头
    table.setLayoutData(gridData);//设置表格布局
    table.setLinesVisible(true);//设置显示表格线
    //创建表头的字符串数组
    String[] tableHeader = {"姓名","性别","电话","电子邮件"};

    for (int i=0;i<tableHeader.length;i++){
        TableColumn tableColumn = new TableColumn(table, SWT.NONE);
        tableColumn.setText( tableHeader[i]);
    }
}

```

```

//添加 3 行数据
TableItem item = new TableItem(table, SWT.NONE);
item.setText(new String[] {"张三", "男", "123", "zhangsan@sina.com"});

item = new TableItem(table, SWT.NONE);
item.setText(new String[] {"李四", "男", "4582", "lisi@sina.com"});

item = new TableItem(table, SWT.NONE);
item.setText(new String[] {"周五", "女", "567", "zhouwu@sina.com"});
//重新布局表格
for (int i=0; i<tableHeader.length; i++) {
    table.getColumn(i).pack ();
}
}

public static void main(String[] args) {
    //调用主窗口
    Display display = Display.getDefault();
    TableSample thisClass = new TableSample();
    thisClass.createSShell();
    thisClass.sShell.open();
    while (!thisClass.sShell.isDisposed()) {
        if (!display.readAndDispatch())
            display.sleep();
    }
    ImageFactory.dispose();
    display.dispose();
}

//创建主窗口
private void createSShell() {
    sShell = new Shell();
    sShell.setText("表格窗口");
    sShell.setLayout(new FillLayout());
    createViewForm();
    sShell.setImage( ImageFactory.loadImage( sShell.getDisplay(), ImageFactory.SAMPLES));
    sShell.pack();
}
}

```

以上程序中应该注意以下几点:

- ❑ 读者可能有些奇怪, 这个示例的程序结构与前面学习的结构不太一样, 这是因为该程序使用了 **JavaBean**, 即把每个控件看作是该类的属性。在实际的项目中, 都是采用 **JavaBean** 的方式, 所以希望读者理解这样的程序结构, 为以后的项目实践打好基础。
- ❑ 该程序中的布局使用了 **ViewForm** 布局, 这个布局适用于放置工具栏。**ViewForm** 的使用方法也很简单, 这里不多作详细解释了。
- ❑ 请读者仔细阅读带有关创建表格方法 **createTable()** 中的代码。

9.9.1 Table、TableItem 和 TableColumn 的关系

一个表格 (Table) 由表头 (TableColumn) 和许多行数据 (TableItem) 组成, 可使用 `setHeaderVisible(boolean show)` 方法来设置是否是显示表头。Table、TableItem 和 TableColumn 的关系可如图 9.45 表示。

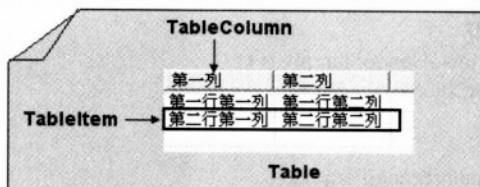


图 9.45 Table、TableItem 和 TableColumn 的关系图

通常创建一个表格的基本步骤如下:

```
// 创建表格对象
Table table = new Table(composite, SWT.HIDE_SELECTION);
//创建两列的表头
TableColumn column1 = new TableColumn(table, SWT.NONE);
column1.setText("第一列");
TableColumn column2 = new TableColumn(table, SWT.NONE);
column2.setText("第二列");
//创建一行数据
TableItem item = new TableItem(table, SWT.NONE);
item.setText(new String[] {"第一行第一列", "第一行第二列"});
//重新布局表格列, 否则需要设置大小
for (int i=0; i<table.getColumnCount(); i++) {
    table.getColumn(i).pack();
}
```

9.9.2 设置带有选择框的表格

上个程序中创建表格时使用了 `SWT.FULL_SELECTION` 样式, 如果将该样式换成 `SWT.CHECK`, 则可以在每一行显示带多选框的表格。将示例程序中创建表格的代码改为如下所示: `table = new Table(composite, SWT.CHECK);`, 代码运行后显示的效果如图 9.46 所示。

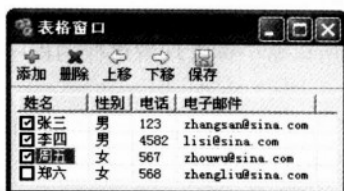


图 9.46 带选择框的表格

此时，单击“删除”按钮后不是删除所选中的行，而是删除选中的所有选项，还要更改以下删除按钮的事件。删除按钮事件更改后的代码如下：

```
//以上代码省略，以下是修改后删除按钮的事件
else if ( event.widget == del)
{
    //首先获得表格中所有的行
    TableItem[] items = table.getItems();
    //循环所有行
    for ( int i=0;i<items.length;i++){
        //如果该行没有被选中，继续循环
        if ( !items[i].getChecked())
            continue;
        //否则选中，查找该表格中是否有该行
        int index = table.indexOf( items[i]);
        //如果没有该行，继续循环
        if (index<0)
            continue;
        //如果有该行，删除该行
        table.remove( index );
    }
}
```

使用带选择框的表格时，需要注意：Table 对象的 getSelection()方法，返回的是选中的 TableItem，而不是选中选择框中的 TableItem。所以要想获得该行是否被选中，需要使用 TableItem 对象的 getChecked()方法，如果返回 true，表示该行被选中；如果返回 false 表示该行未被选中。

9.9.3 设置可同时选中多行表格

默认情况下创建的表格是 SWT.SINGLE 样式，即只能选中一行，而要使表格可以同时选中多行，可以使用 SWT.MULTI 样式。例如，创建可以选择多行的表格代码如下：

```
Table table = new Table(composite, SWT.MULTI|SWT.FULL_SELECTION);
```

可以多行选中，并使用同时选择一行的样式效果如图 9.47 所示。

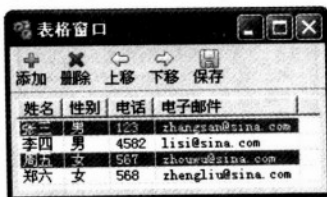


图 9.47 可多行显示的表格

下面总结一下表格 (Table) 可用的样式：

□ SWT.SINGLE 或 SWT.MULTI：指定单选还是可以同时多选几行。

- ❑ SWT.CHECK: 显示带有选择框的表格。
- ❑ SWT.FULL_SELECTION: 当鼠标选中某一行时, 整行高亮显示。
- ❑ SWT.VIRTUAL: 支持大型表(特定于平台)。

对于表头(TableColumn)也有以下可用的样式:

- ❑ SWT.LEFT: 靠左对齐, 默认值。
- ❑ SWT.RIGHT: 靠右对齐。
- ❑ SWT.CENTER: 居中对齐。

9.9.4 可拖动的表格

通常情况下, 由于每个用户的使用习惯不同, 会按照自己的喜好来设置表每列的位置, 一般是通过拖动表头的办法来实现的。如图 9.48 所示为一个可随意拖动表格的示意图。



图 9.48 可拖动表头的表格

SWT 中提供了很简单的方法来实现该功能, 只要在创建 TableColumn 时通过 setMoveable(boolean moveable)方法, 将是否可移动的属性设置为 true 即可。例如, 将上面程序中创建表头时的代码改为如下:

```
String[] tableHeader = {"姓名", "性别", "电子邮件", "电话"};
for (int i=0; i<tableHeader.length; i++){
    TableColumn tableColumn = new TableColumn(table, SWT.NONE);
    tableColumn.setText(tableHeader[i]);
    //设置表头可移动, 默认为 false
    tableColumn.setMoveable(true);
}
```

9.9.5 设置单元格的图标

有时, 为了使表格更加美观, 通常会为单元格添加图标。例如下面的程序中, 在表格的“性别”一栏中, 如果为“男”, 则显示小王子的图标; 如果为“女”, 则显示小公主的图标。设置图标后的表格效果如图 9.49 所示。

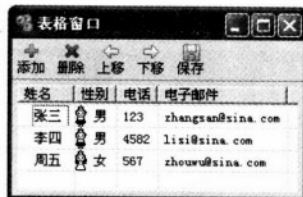


图 9.49 带图标的单元格

设置图标方法不难, 在创建 TableItem 时, 使用创建图标的方法即可。创建图标的方法有以下几种:

- ❑ setImage(Image image): 默认情况下为第一列设置图标。

- ❑ `setImage(Image[] images)`: 为每一列设置图标。
- ❑ `setImage(int index, Image image)`: 为指定索引列设置图标。例如, 为第 2 列设置图标的代码为 `item.setImage(1, new Image(display, "C:\\\\icon.gif"))`。

图 9.49 所示的表格使用的是第 3 种方法, 设置指定索引来设置单元格的图片。添加图标后的代码如下:

```
//添加 3 行数据
TableItem item = new TableItem(table, SWT.NONE);
item.setText(new String[] {"张三", "男", "123", "zhangsan@sina.com"});
//设置图标
item.setImage( ImageFactory.loadImage( table.getDisplay(), ImageFactory.ICON_BOY));
item = new TableItem(table, SWT.NONE);
item.setText(new String[] {"李四", "男", "4582", "lisi@sina.com"});
//设置图标
item.setImage( ImageFactory.loadImage( table.getDisplay(), ImageFactory.ICON_BOY));
item = new TableItem(table, SWT.NONE);
item.setText(new String[] {"周五", "女", "567", "zhouwu@sina.com"});
//设置图标
item.setImage( ImageFactory.loadImage( table.getDisplay(), ImageFactory.ICON_GIRL));
```

9.9.6 改变选中行高亮显示的颜色

当选中某一行数据时, 为了能突出显示出所选的一行数据, 可以改变这一行的文字颜色和单元格的背景色。如图 9.50 所示, 当选中某一行时, 这时将选中的一行显示为红色。

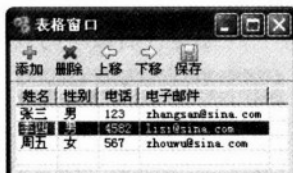


图 9.50 改变选中一行的颜色

`TableItem` 中设置单元格颜色的方法有以下几种:

- ❑ `setBackground(Color color)`: 设置某一行的背景色。
- ❑ `setBackground(int index, Color color)`: 设置指定索引单元格的背景色。
- ❑ `setForeground(Color color)`: 设置某一行的前景色, 通常为文字的颜色。
- ❑ `setForeground(int index, Color color)`: 设置指定索引值的前景色。

要实现单击每一行改变颜色, 光有设置颜色的方法是不够的, 要配合表格的单击事件来实现。例如这里示例程序的原理是, 当单击某一行时, 循环所有的行, 然后判断该行是否选中, 通过 `isSelected(int index)` 方法来判断, 如果该行选中, 则改变背景色和前景色, 否则取消设置的背景色。该程序最终实现的代码如下:

```
table.addSelectionListener( new SelectionAdapter(){
    public void widgetSelected(SelectionEvent e) {
```

```

//获得所有的行数
int total = table.getItemCount();
//循环所有行
for ( int i=0;i<total;i++){
    TableItem item = table.getItem(i);
    //如果该行为选中状态, 改变背景色和前景色, 否则设置颜色
    if (table.isSelected( i )){
        item.setBackground( sShell.getDisplay().getSystemColor( SWT.COLOR_RED));
        item.setForeground( sShell.getDisplay().getSystemColor( SWT.COLOR_WHITE));
    }else {
        item.setBackground( null );
        item.setForeground( null );
    }
}
});

```

9.9.7 带有上下文菜单的表格

一般出于界面友好性的角度考虑, 在开发时要使用上下文菜单 (Context Menu)。上下文菜单是当用户在某一个特定的场合可以使用的操作。以 Eclipse 工作区为例, 如果在编辑区的不同区域单击鼠标右键, 弹出的快捷菜单是不一样的, 不同的菜单项根据当前鼠标所在区域的不同, 可进行的操作也不同。

下面就为 9.9.6 节中例子的程序添加上下文菜单, 当在所选的单元格上右击时, 弹出可以进行删除或查看操作的快捷菜单。程序运行后最终的效果图如图 9.51 所示。

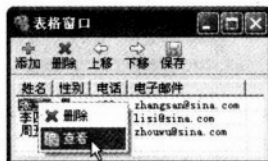


图 9.51 带有上下文菜单的表格

该程序的实现原理是, 创建一个类型为 SWT.POP_UP 的菜单 (Menu), 然后通过 Table 对象的 setMenu(Menu menu)方法, 为该表格设置菜单。修改后的程序如下:

(1) 在 TableSample 类中添加一个菜单属性:

```
private Menu menu = null;
```

(2) 在创建窗口的 createSShell()方法中添加 createMenu()方法:

```

private void createSShell() {
    //前面程序省略
    createMenu();
    //后面程序也省略
}

```


(3) 添加创建菜单的 createMenu()方法:

```
//创建上下文菜单
private void createMenu() {
    //创建弹出式菜单
    menu = new Menu (sShell, SWT.POP_UP);
    //设置该菜单为表格菜单
    table.setMenu (menu);
    //创建删除菜单项
    MenuItem del = new MenuItem (menu, SWT.PUSH);
    del.setText ("删除");
    del.setImage(ImageFactory.loadImage( sShell.getDisplay(), ImageFactory.DELETE_
EDIT));
    //为删除菜单注册事件, 当单击时, 删除所选择的行
    del.addListener (SWT.Selection, new Listener () {
        public void handleEvent (Event event) {
            table.remove (table.getSelectionIndices ());
        }
    });
    //创建查看菜单项
    MenuItem view = new MenuItem (menu, SWT.PUSH);
    view.setText ("查看");
    view.setImage(ImageFactory.loadImage( sShell.getDisplay(), ImageFactory.SCOPY_
EDIT));
    //为查看菜单项注册事件, 当单击时打印出所选的姓名
    view.addListener (SWT.Selection, new Listener () {
        public void handleEvent (Event event) {
            TableItem[] items = table.getSelection();
            for (int i=0;i<items.length;i++)
                System.out.print(items[i].getText());
        }
    });
}
```

9.9.8 可编辑的表格 (TableEditor)

之前使用的表格都是只能够查看, 并不能够修改, 那能不能直接在表格中修改数据呢? 答案是肯定的。但需要结合 org.eclipse.swt.custom.TableEditor 类来使用, 如图 9.52 和图 9.53 所示。添加修改功能后, 为第一列添加文本框, 使用户可以直接输入姓名, 为第二列添加下拉按钮, 使用户可以选择性别。

要实现该功能, 需要在创建表格的 createTable()方法中, 在初始化表格数据后, 添加以下代码:

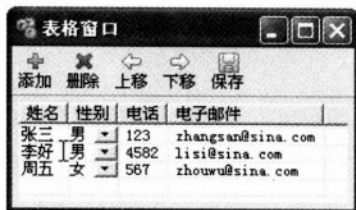


图 9.52 可直接输入姓名

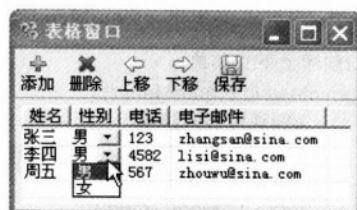


图 9.53 可直接选择性别

//添加可编辑的单元格

```
TableItem[] items = table.getItems();
for (int i=0; i<items.length; i++) {
    //第一列设置, 创建 TableEditor 对象
    final TableEditor editor = new TableEditor (table);
    //创建一个文本框, 用于输入文字
    final Text text = new Text (table, SWT.NONE);
    //将文本框当前值设置为表格中的值
    text.setText(items[i].getText(0));
    //设置编辑单元格水平填充
    editor.grabHorizontal = true;
    //关键方法, 将编辑单元格与文本框绑定到表格的第一列
    editor.setEditor(text, items[i], 0);
    //当文本框改变值时, 注册文本框改变事件, 该事件改变表格中的数据
    //否则即使改变文本框的值, 对表格中的数据也不会影响
    text.addModifyListener( new ModifyListener(){
        public void modifyText(ModifyEvent e) {
            editor.getItem().setText(1,text.getText());
        }
    });
    //同理, 为第二列绑定下拉框 CCombo
    final TableEditor editor1 = new TableEditor (table);
    final CCombo combo = new CCombo (table, SWT.NONE);
    combo.add("男");
    combo.add("女");
    combo.setText(items[i].getText(1));
    editor1.grabHorizontal = true;
    editor1.setEditor(combo, items[i], 1);
    combo.addModifyListener( new ModifyListener(){
        public void modifyText(ModifyEvent e) {
            editor1.getItem().setText(1,combo.getText());
        }
    });
}
```

在使用 TableEditor 对象时应注意以下几个问题:

□ TableEditor 对象通过使用 setEditor(Control editor, TableItem item, int column)方法, 将

一个控件与某一个单元格绑定。当然除了上述程序中绑定了文本框 (Text) 和下拉按钮 (CCombo) 外, 也可以设置任何的 Control 控件, 如按钮等。

- 虽然为单元格绑定了相对应的编辑控件, 但控件和表格之间是独立的, 要想改变编辑控件就改变了表格中的数据, 还需要配合编辑控件的事件来处理。例如在上面的程序中, 添加了改变控件事件, 一旦某个控件的值改变, 也改变表格中对应单元格的数据。

9.9.9 用键盘控制表格 (TableCursor)

以上所创建的表格只能通过鼠标来进行选择, 一些用户习惯用键盘来进行选择, 那能不能实现呢? 通过 org.eclipse.swt.custom.TableCursor 类就可以为表格添加键盘事件, 例如在创建完表格的代码中使用以下代码:

```
TableCursor cursor = new TableCursor(table, SWT.NONE);
```

就可以让键盘的上下左右键来控制光标的移动了。当然, 如何配合其他的事件就能使表格的功能更强大呢?

现在要实现这样一个程序, 功能如下: 利用键盘移动光标到任意单元格, 当在单元格上按 Enter 键时, 使该单元格处于可编辑状态。当编辑完成后, 按 Enter 键保存; 如果按 Esc 键, 则取消当前的修改。这样, 就可以在表格的任意单元格进行直接编辑了。该程序运行后效果图如图 9.54 所示。

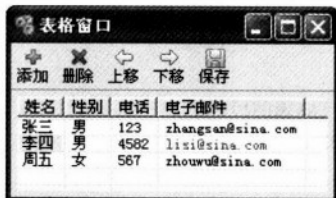


图 9.54 使用键盘控制的表格

实现该功能的最终代码如下:

```
//创建 TableCursor 对象, 使用上下左右键可以控制表格
final TableCursor cursor = new TableCursor(table, SWT.NONE);
//创建可编辑的控件
final ControlEditor editor = new ControlEditor(cursor);
editor.grabHorizontal = true;
editor.grabVertical = true;
//为 TableCursor 对象注册事件
cursor.addSelectionListener( new SelectionAdapter() {
    //移动光标, 在单元格上按 Enter 键所触发的事件
    public void widgetDefaultSelected(SelectionEvent e) {
        //创建一个文本框控件
        final Text text = new Text(cursor, SWT.NONE);
        //获得当前光标所在的行 TableItem 对象
```

```

        TableItem row = cursor.getRow();
        //获得当前光标所在的列数
        int column = cursor.getColumn();
        //当前光标所在单元格的值赋给文本框
        text.setText(row.getText(column));
        //为文本框注册键盘事件
        text.addKeyListener(new KeyAdapter() {
            public void keyPressed(KeyEvent e) {
                //此时在文本框上按 Enter 键后，这时表格中的数据为修改后文本框中的数据，
                然后释放文本框资源
                if (e.character == SWT.CR) {
                    TableItem row = cursor.getRow();
                    int column = cursor.getColumn();
                    row.setText(column, text.getText());
                    text.dispose();
                }
                //如果在文本框上按 Esc 键，则并不对表格中的数据进行修改
                if (e.character == SWT.ESC) {
                    text.dispose();
                }
            }
        });
        //注册焦点事件
        text.addFocusListener(new FocusAdapter() {
            //当该文本框失去焦点时，释放文本框资源
            public void focusLost(FocusEvent e) {
                text.dispose();
            }
        });
        //将该文本框绑定到可编辑的控件上
        editor.setEditor(text);
        //设置文本框的焦点
        text.setFocus();
    }
    //移动光标到一个单元格上所触发的事件
    public void widgetSelected(SelectionEvent e) {
        table.setSelection(new TableItem[] { cursor.getRow()});
    }
}
};

```

该程序使用时应注意以下几点：

- ❑ 该程序的基本原理是：当对单元格进行编辑时，其实是对一个文本框进行编辑。使用 `ControlEditor` 对象将文本框和单元格绑定起来。当完成对文本框编辑后，再更改表格中的值为文本框的值。
- ❑ 在 `TableCursor` 对象事件处理中，`widgetDefaultSelected(SelectionEvent e)` 方法是当光标移动到单元格上按 Enter 键触发的，而 `widgetSelected(SelectionEvent e)` 方法是当光标移动到单元格上就触发的。

- ❑ 对文本框编辑的过程中使用的键盘事件。即按 Enter 键事件和按 Esc 键事件, 判断是按下了哪个按键的方法是 `if (e.character == SWT.CR)`。

9.9.10 带有进度条的表格

在一些下载软件中, 经常可以看到一个单元格显示某个文件下载进度的情况。SWT 的表格也可以实现这样的功能。如图 9.55 所示即为一个带有进度条的表格。

创建这样一个表格的代码如下:



图 9.55 带进度条的表格

ProgressBarTableSample.java

```
// 创建表格对象
Table table = new Table(shell, SWT.BORDER);
table.setHeaderVisible(true);
table.setLinesVisible(true);
// 创建表头
for (int i = 0; i < 2; i++) {
    new TableColumn(table, SWT.NONE);
}
table.getColumn(0).setText("任务");
table.getColumn(1).setText("进度");
// 创建 10 行数据
for (int i = 0; i < 10; i++) {
    //创建一行
    TableItem item = new TableItem(table, SWT.NONE);
    item.setText("任务 " + i);
    //创建一个进度条
    ProgressBar bar = new ProgressBar(table, SWT.NONE);
    bar.setSelection((i + 1) * 10);
    //创建一个可编辑的表格对象
    TableEditor editor = new TableEditor(table);
    editor.grabHorizontal = true;
    editor.grabVertical = true;
    //将进度条绑定到单元格上
    editor.setEditor(bar, item, 1);
}
table.getColumn(0).pack();
table.getColumn(1).setWidth(100);
table.pack();
```

该程序的实现需要注意以下几个问题:

- ❑ 其实这个程序实现起来并不复杂, 在单元格中显示进度条是通过 `TableEditor` 对象来实现的, 在 9.9.8 节中已经接触到该类, 在这里笔者想着重说明的是, 通过 `TableEditor` 对象可以将任何控件 (`Control` 类及其子类) 绑定到单元格中, 只要读

者有足够的想象力，并且配合事件的处理，便可以编写出功能强大的表格。

- ❑ 该示例的程序中，虽然使用了进度条，但并不是动态的，要想使进度条动起来要用到线程的知识。

9.9.11 表格小结

表格是很重要的控件，所以花了大量的篇幅来介绍它，但在实际的项目中，通常使用 JFace 中的表格 `org.eclipse.jface.viewers.TableViewer` 类。`TableViewer` 将数据和视图分开，使用了 MVC 的设计模式，在 JFace 篇还要详细说明。

笔者一贯认为，打好基础是很重要的事情，只有掌握了基础知识，才能在更高的层次上游刃有余。所以即使在以后的使用中可能很少遇到 SWT 中表格的使用，也需要了解最基本的表格操作，希望读者能够掌握其中的原理。

9.10 树 (Tree)

树是一种非常重要的数据结构，通常用于分层显示数据，例如操作系统的文件目录就是一个树结构。如图 9.56 所示为一个树的效果示意图。

从图中可以看出一个树 (Tree) 由多个 `TreeItem` 组成，树展开后每一行都可以看作是一个 `TreeItem`。创建这样一个树形结构的控件需要以下几个步骤：

(1) 首先创建一个树对象，该树的样式是只能单选，并且是带边框的树。

```
Tree tree = new Tree(shell, SWT.BORDER | SWT.SINGLE);
```

(2) 创建根节点的 `TreeItem`，这里使用的是 `TreeItem` 的构造方法 `TreeItem (Tree parent, int style)`。

```
TreeItem root = new TreeItem( tree , SWT.NULL);
root.setText("根节点");
```

(3) 创建位于根节点下的 `TreeItem`，也就是第一层的 3 个节点：

```
TreeItem child1 = new TreeItem ( root , SWT.NULL);
child1.setText("子孙 1");
TreeItem child2 = new TreeItem ( root , SWT.NULL);
child2.setText("子孙 2");
TreeItem child3 = new TreeItem ( root , SWT.NULL);
child3.setText("子孙 3");
```

这里使用的是 `TreeItem` 的构造方法 `TreeItem (TreeItem parentItem, int style)`。

(4) 同理，为“子孙 1”节点添加子孙：

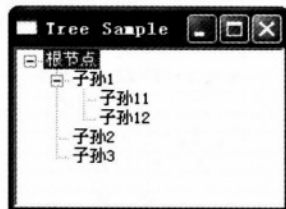


图 9.56 树示意图

```

TreeItem child11 = new TreeItem ( child1 , SWT.NULL);
child11.setText("子孙 11");
TreeItem child12 = new TreeItem ( child1 , SWT.NULL);
child12.setText("子孙 12");

```

9.10.1 不同样式的树

除了上述程序使用了 SWT.SINGLE 样式外，还有其他树的样式：

- ❑ SWT.MULTI：可多选树，按 Ctrl 键可同时选择多个。
- ❑ SWT.CHECK：可带有选择框的树，如图 9.57 所示。
- ❑ SWT.FULL_SELECTION：选中一项时，将高亮显示整行，如图 9.58 所示。

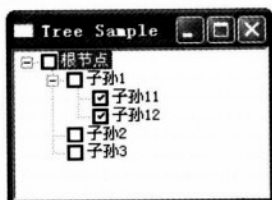


图 9.57 带选择框的树 (SWT.CHECK)

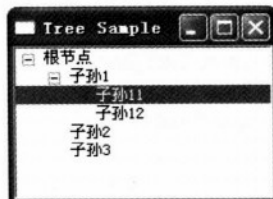


图 9.58 SWT.FULL_SELECTION 样式的树

9.10.2 为树添加图标

为了使树的外观更加人性化，通常要配合图标的使用。如图 9.59 所示为带有图标的树结构。

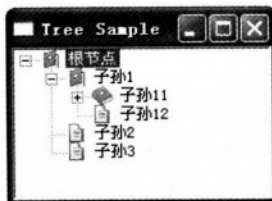


图 9.59 带图标的树

实现代码如下：

TreeSample.java

```

package com.fengmanfei.ch9;

import org.eclipse.swt.SWT;
import org.eclipse.swt.events.*;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.widgets.*;
import com.fengmanfei.util.ImageFactory;

```



```

public class TreeSample {
    public static void main(String[] args) {
        final Display display = new Display();
        final Shell shell = new Shell(display);
        shell.setText("Tree Sample");
        shell.setLayout(new FillLayout());
        //创建一个树对象
        final Tree tree = new Tree(shell, SWT.BORDER | SWT.SINGLE);
        //创建树的一个根节点
        TreeItem root = new TreeItem( tree , SWT.NULL);
        root.setText("根节点");
        //创建子孙节点
        TreeItem child1 = new TreeItem ( root , SWT.NULL);
        child1.setText("子孙 1");
        TreeItem child2 = new TreeItem ( root , SWT.NULL);
        child2.setText("子孙 2");
        TreeItem child3 = new TreeItem ( root , SWT.NULL);
        child3.setText("子孙 3");

        TreeItem child11 = new TreeItem ( child1 , SWT.NULL);
        child11.setText("子孙 11");
        TreeItem child12 = new TreeItem ( child1 , SWT.NULL);
        child12.setText("子孙 12");

        TreeItem child111 = new TreeItem ( child11 , SWT.NULL);
        child111.setText("子孙 111");
        TreeItem child112 = new TreeItem ( child11 , SWT.NULL);
        child112.setText("子孙 112");
        //调用 convertImage 方法来设置树的图标
        convertImage( tree );
        //为树注册树监听事件
        tree.addTreeListener( new TreeListener(){
            //当折叠树节点时
            public void treeCollapsed(TreeEvent e) {
                //首先获得触发事件的 TreeItem
                TreeItem item = (TreeItem)e.item;
                //将该节点的图标设置为关闭状态
                item.setImage(ImageFactory.loadImage(tree.getDisplay(),
ImageFactory.TOC_CLOSED));
            }
            //当展开树节点时
            public void treeExpanded(TreeEvent e) {
                TreeItem item = (TreeItem)e.item;
                item.setImage(ImageFactory.loadImage(tree.getDisplay(),
ImageFactory.TOC_OPEN));
            }
        });
    }
}

```






```

shell.setSize( 200,150);
shell.open();
while (!shell.isDisposed()) {
    if (!display.readAndDispatch())
        display.sleep();
}
ImageFactory.dispose();
display.dispose();
}
//设置树图标的方法
public static void convertImage ( Tree tree ){
    //这里假设只有一个根节点
    TreeItem[] items = tree.getItems();
    //首先根据根节点的状态设置图标
    if (items[0].getExpanded())//如果该节点为展开状态
        items[0].setImage(ImageFactory.loadImage(tree.getDisplay(),
ImageFactory.TOC_OPEN));
    else//否则, 如果为折叠状态
        items[0].setImage(ImageFactory.loadImage(tree.getDisplay(),
ImageFactory.TOC_CLOSED));
    //设置该根节点的图标
    setChildImage ( items[0] );
}
//设置一个节点的方法, 该方法非常重要, 要理解该方法的递归用法
//参数 item 可以单独看作是树中的某一个 TreeItem
public static void setChildImage ( TreeItem item ){
    //首先获得该 TreeItem 的所有子 TreeItem
    TreeItem[] items = item.getItems();
    //循环每一个 TreeItem
    for( int i=0;i<items.length;i++)
    {
        //如果这个 TreeItem 下没有子孙
        if (items[i].getItems().length==0)
            items[i].setImage(ImageFactory.loadImage(item.getDisplay(),
ImageFactory.TOPIC));
        else{//如果这个 TreeItem 有多个子孙
            //如果这个 TreeItem 是展开状态, 则设置展开的图片
            if ( items[i].getExpanded())
                items[i].setImage(ImageFactory.loadImage(item.getDisplay(),
ImageFactory.TOC_OPEN));
            else //否则, 则设置折叠的图片
                items[i].setImage(ImageFactory.loadImage(item.getDisplay(),
ImageFactory.TOC_CLOSED));
            //要为该 TreeItem 的子孙设置图标, 递归调用 setChildImage 方法
            setChildImage ( items[i] );
        }
    }
}
}
}

```

通过该程序可以总结出使用树时的一些注意事项：

- ❑ 本程序的思路是，在创建完树的数据后，调用 `convertImage(tree)` 方法，根据每个 `TreelItem` 的状态设置树的图标；然后为树注册 `TreeListener` 事件，即设置展开和折叠 `TreelItem` 所触发的事件，当展开和折叠节点时，根据状态改变节点的图标。
- ❑ 一个 `TreelItem` 一共有 3 种状态：
 - ◆ ：该 `TreelItem` 没有子孙时，既不能展开也不能折叠，是最简单的状态。
 - ◆ ：该 `TreelItem` 有子孙时，并且该节点的状态为折叠状态。通过 `getExpanded() == false` 方法来判断。
 - ◆ ：该 `TreelItem` 有子孙时，并且该节点的状态为展开状态。通过 `getExpanded() == true` 方法来判断。
- ❑ 理解了一个 `TreelItem` 的性质，就可以理解 `setChildImage` 方法了，该方法把传递过来的参数 `item` 看作是任意的一个 `TreelItem`。难点是当该 `TreelItem` 有多个子孙时要递归调用 `setChildImage` 方法。递归是树的一个重要的性质，如果读者仍有困惑，可以阅读一些有关数据结构方面的书籍。
- ❑ 对树展开与折叠事件的处理也是使用树时应该注意的问题。请读者参考代码的注释部分。

9.10.3 可编辑的树

前面所介绍程序的树，还只是能够浏览，并不能直接对树进行修改，那可不可以对树的节点进行修改呢？答案是肯定的，在前面学习表格时，知道了结合使用 `TableEditor` 可以使表格具有修改的功能，相应的树也有 `TreeEditor` 类，该类在 `org.eclipse.swt.custom` 包下。如图 9.60 所示，即为一个可编辑的树。当在节点上双击时，可直接编辑节点数据。



图 9.60 可编辑的树

实现该代码的程序需要在 `TreeSample.java` 源代码中直接修改，添加的代码如下：

```
//创建一个可编辑的 TreeEditor 对象
final TreeEditor editor = new TreeEditor(tree);
editor.horizontalAlignment = SWT.LEFT;
editor.grabHorizontal = true;
editor.minimumWidth = 30;
//注册选中事件
tree.addSelectionListener(new SelectionAdapter() {
    //当鼠标双击节点时使节点可编辑
    public void widgetDefaultSelected(SelectionEvent e) {
        //释放之前编辑的控件
        Control oldEditor = editor.getEditor();
        if (oldEditor != null) oldEditor.dispose();
        //获得触发事件的 TreelItem，如果为 null，返回
        TreelItem item = (TreelItem)e.item;
```

```

if (item == null) return;
//创建一个文本框，作为编辑节点时输入的文字
Text newEditor = new Text(tree, SWT.NONE);
//将树节点的值赋值给文本框
newEditor.setText(item.getText());
//当文本框的值改变时，也相应地改变树节点数据的值
newEditor.addModifyListener(new ModifyListener() {
    public void modifyText(ModifyEvent e) {
        Text text = (Text)editor.getEditor();
        editor.getItem().setText(text.getText());
    }
});
newEditor.selectAll();//选中所有文本框
newEditor.setFocus();//并将焦点设置为该文本框
//将树节点与文本框节点绑定
editor.setEditor(newEditor, item);
}
});

```

该程序的实现原理大致如下：

在编辑树节点时，实际上是在编辑一个文本框，当改变文本框的值后，调用文本框的文本改变事件，将文本框的值赋给对应的树的节点，而将文本框与树节点联系起来的是 TreeEditor 对象。可以理解为 TreeEditor 对象为一个中介，绑定文本框和某个节点。当然，这个程序中绑定的是文本框，也可以绑定其他的控件，例如按钮、下拉框等。凡是 Control 类与其子类都可以与节点绑定，这就使树有了很大的灵活性。

9.10.4 表格树

还有一种特殊的树——表格树。表格树综合了数和表格的特性，可以为树增加表头。如图 9.61 所示为带有表头的表格树。

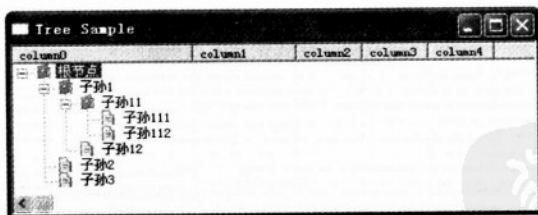


图 9.61 表格树

创建这样一个表格树需要在以上程序的基础上添加如下代码：

```

//为树创建 5 列
for (int i=0; i<5; i++){
    TreeColumn column = new TreeColumn(tree, SWT.NONE);
    column.setText("column"+i);
}

```

```
for ( int i=0;i<tree.getColumnCount();i++)  
tree.getColumn(i).pack();  
//设置网格线可见  
tree.setLinesVisible( true );  
//设置表头可见  
tree.setHeaderVisible( true );
```

在 Eclipse 3.1 之前的版本中, SWT 提供了专门的表格树类(`TableTree` 和 `TableTreeItem`)。但在 Eclipse 3.1 以后, 只需用为 `Tree` 设置 `TreeColumn` 就可以达到与表格树类相同的效果, 表格树比较简单, 这里不多作详细介绍了。

9.10.5 树小结

通过本小节对树的学习, 读者可能会发现树和表格有许多相通之处, 但树有一个最重要的性质——递归, 希望读者能够深刻体会。另外, 在实际的项目开发中, 通常不会使用 SWT 的树, 而是使用 JFace 中的 `org.eclipse.jface.viewers.TreeViewer` 类, `TreeViewer` 与 `TableViewer` 一样, 将数据和视图分开, 功能更加强大。有关 `TreeViewer` 的学习将在 JFace 篇进行讲述。

9.11 格式化文本 (`StyleText`)

格式化文本是可以将文本格式化的对象, 例如在 Word 中, 可以设置文字的大小、字体, 选择文字的颜色, 设置字符是否加粗等。这些都是将文本格式化后的效果。SWT 中格式化文本 (`StyleText`) 就提供了与之类似的功能。如图 9.62 所示, 即为一个格式化后的文本框。

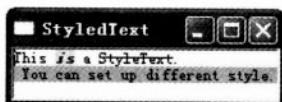


图 9.62 格式化文本效果图

格式化文本通常可以对文本格式化以下内容:

- ☐ 文本的前景色 (`foreground`) 和背景色 (`background`) 。
- ☐ 文本的字体样式: 普通 (`SWT.NORMAL`)、加粗 (`SWT.BOLD`) 或是倾斜 (`SWT.ITALIC`); 是否带下划线 (`underline`); 是否带删除线 (`strikeout`) 。
- ☐ 文本一行的背景色等。

了解了格式化文本 (`StyleText`) 的特性, 就能理解用 `StyleText` 对象的用途了。例如说 Eclipse Java 编辑器就是一个功能强大的格式化文本。它可以为一些关键字自动着色, 并能够在输入一些字符时自动提示。事实上, Eclipse 使用的编辑器正是 `StyleText` 的扩展, JFace 扩充 SWT 的很大一部分功能就是提供了对格式化文本的支持。这些类可以在 JFace 的包 `org.eclipse.jface.text` 中找到。Eclipse 核心的平台也是利用了 JFace 对文本的支持。

9.11.1 格式化对象 (StyleRange)

将文本设置为不同的样式是使用 StyleRange 对象来设置的。首先看一下如图 9.62 所示的程序是如何实现的。代码如下：

StyleTextSample.java

```
//创建一个 StyledText 对象
StyledText styleText = new StyledText( shell, SWT.BORDER|SWT.WRAP|SWT.MULTI);
styleText.setText("This is a StyleText.\n You can set up different style.\n");
//创建一个 StyleRange 对象, 该对象将格式化文本中的"is"
StyleRange styleRange1 = new StyleRange();
styleRange1.start = 5;//设置使用该样式开始的字符
styleRange1.length = 2;//设置使用该样式字符的长度
//设置样式的前景色和背景色
styleRange1.foreground = shell.getDisplay().getSystemColor(SWT.COLOR_BLUE);
styleRange1.background = shell.getDisplay().getSystemColor(SWT.COLOR_YELLOW);
styleRange1.fontStyle = SWT.ITALIC|SWT.BOLD;//设置字体为倾斜并且加粗

//设置另一个 StyleRange 对象, 该对象将格式化文本中的"StyleText"
StyleRange styleRange2 = new StyleRange();
styleRange2.start = 10;
styleRange2.length = 9;
styleRange2.fontStyle = SWT.NORMAL;
styleRange2.strikeout=true;

//将两个文本样式应用到 StyledText 对象
styleText.setStyleRange(styleRange1);
styleText.setStyleRange(styleRange2);

//设置第二行的文字的 background 为灰色
styleText.setLineBackground(1, 1, shell.getDisplay().getSystemColor(SWT.COLOR_GRAY));
```

下面详细理解一下 StyledText 对象与 StyleRange 对象之间的关系。一个 StyleRange 代表一种文本的样式。可以将这种样式设置为 StyledText 中的任何文本，它们之间的关系如图 9.63 所示。

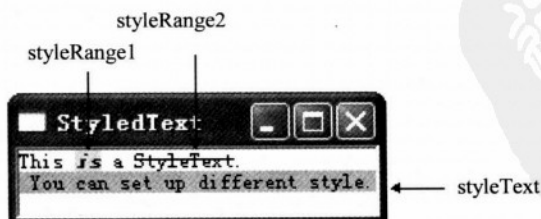


图 9.63 StyleRange 与 StyledText 的关系示意图

创建和使用 `StyleRange` 对象的步骤如下：

(1) 创建一个 `StyleRange` 对象。

```
StyleRange styleRange1 = new StyleRange();
```

(2) 设定将该样式作用于 `StyledText` 位置。例如，上例中的代码如下：

```
styleRange1.start = 5;//设置使用该样式开始的字符  
styleRange1.length = 2;//设置使用该样式字符的长度
```

其中 `start` 属性是开始字符的位置，`length` 从开始字符位置截取两个字符。例如，如下的这段文本中：

```
styleText.setText("This is a StyleText.\n You can set up different style.\n");
```

- ☐ `start = 5, length = 2` 取出的是字符 “is”。
- ☐ `start = 11, length = 9` 取出的是字符 “StyleText”。
- (3) 设置样式的其他属性：
 - ☐ `background`: 背景色。
 - ☐ `foreground`: 前景色。
 - ☐ `fontStyle`: 字体样式，如 `SWT.NORMAL`、`SWT.ITALIC` 或 `SWT.BOLD`。
 - ☐ `strikeout`: 是否带删除线，`true` 为显示删除线，`false` 为不显示删除线。
 - ☐ `underline`: 是否带下划线，`true` 为显示下划线，`false` 为不显示下划线。

9.11.2 格式化文本的事件处理

理解了如何设置文本的样式是不够的，要配合 `StyledText` 对象的事件处理，才能更深入理解格式化文本的强大功能。`StyledText` 可注册的事件有以下几种：

(1) 有文本选择事件 (Selection Event)：当选择一段文本后所触发的事件。

(2) 有关改变文本所涉及的事件：

- ☐ 键盘更改事件 (`VerifyKey Event`)：当按下键盘时触发的事件。
- ☐ 文本更改事件 (`Verify Event`)：当文本改变时触发的事件。
- ☐ 文本更改后事件 (Modify Event)：当文本改变后所触发的事件，一般用于在文件保存时提示用户是否保存。
- ☐ 扩展的文本改变后事件 (ExtendedModify Event)：比 `Modify Event` 携带更多的有关文本改变情况的信息。

(3) 有关对一行文本改变时触发的事件：

- ☐ 设置行背景色事件 (`LineGetBackground Event`)：可设置行的背景色。
- ☐ 设置行样式事件 (`LineGetStyle Event`)：可设置行的字体样式。

这里列举的事件的具体用法将在以下几节中结合实例来进行讲解。

9.11.3 对选中文本设置格式

首先看一下文本选择事件（Selection Event）如何使用，设计如下的一小程序，该程序的功能是，对选中的文本可以设置样式，就像 Word 中对文本的操作一样。程序运行后如图 9.64 所示。当选中一段文本后，可单击工具栏的编辑按钮来改变所设置的样式。

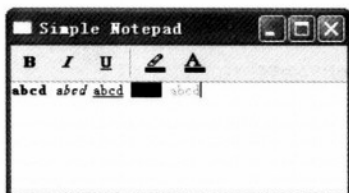


图 9.64 简单文本编辑器

该程序也是由一个工具栏和一个可格式化的文本框 `StyledText` 组成，布局采用了 `ViewForm`。程序功能实现关键是对工具栏几个按钮的事件处理上，以下只显示创建工具栏的方法，完整的代码请参阅本书附带的光盘。

SimpleNotepad.java

```
private void createToolBar() {
    //创建工具栏和工具栏按钮
    toolBar = new ToolBar(viewForm, SWT.FLAT);
    final ToolItem boldButton = new ToolItem(toolBar, SWT.PUSH);
    boldButton.setImage( ImageFactory.loadImage(toolBar.getDisplay(), ImageFactory.BOLD));
    final ToolItem italicButton = new ToolItem(toolBar, SWT.PUSH);
    italicButton.setImage( ImageFactory.loadImage(toolBar.getDisplay(), ImageFactory.ITALIC));
    final ToolItem underlineButton = new ToolItem(toolBar, SWT.PUSH);
    underlineButton.setImage( ImageFactory.loadImage(toolBar.getDisplay(), ImageFactory.
UNDERLIN));
    new ToolItem(toolBar, SWT.SEPARATOR);
    final ToolItem bgColorButton = new ToolItem(toolBar, SWT.PUSH);
    bgColorButton.setImage( ImageFactory.loadImage(toolBar.getDisplay(), ImageFactory.
BGCOLOR));
    final ToolItem forColorButton = new ToolItem(toolBar, SWT.PUSH);
    forColorButton.setImage( ImageFactory.loadImage(toolBar.getDisplay(), ImageFactory.
FORCOLOR));
    //创建按钮事件处理对象
    Listener listener = new Listener(){
        public void handleEvent(Event event) {
            //如果当前未选中文本返回，不处理任何事件
            if (styledText.getSelectionCount()==0)
                return;
            //声明一个 StyleRange 对象
            StyleRange styleRange=null;
            //获得当前所选择文本所在的开始位置和长度
```

```

        Point select = styledText.getSelectionRange();
        //首先要查找一下当前所选中文本中是否已经有设置过的 StyleRange 对象
        StyleRange[] ranges = styledText.getStyleRanges( select.x,select.y);
        //如果找到了, 则将该选中字符串的第一个 StyleRange 样式作为当前所要改变的样
        式, 否则创建一个新样式
        if( ranges.length>0)
            styleRange=ranges[0];
        else
            styleRange = new StyleRange();
        //设置样式所作用的位置为选择文本的位置
        styleRange.start=select.x;
        styleRange.length=select.y;
        //如果单击加粗按钮, 则要设置字体样式为 SWT.BOLD
        if (event.widget==boldButton)
            styleRange.fontStyle=styleRange.fontStyle|SWT.BOLD;
        //如果单击倾斜按钮, 则要设置字体样式为 SWT.ITALIC
        else if (event.widget==italicButton)
            styleRange.fontStyle=styleRange.fontStyle|SWT.ITALIC;
        //如果单击加下划线按钮, 则要设置 underline 属性为 true
        else if (event.widget==underlineButton)
            styleRange.underline=true;
        //如果单击设置背景色按钮, 则要设置 background 属性
        else if (event.widget==bgColorButton){
            ColorDialog dialog = new ColorDialog(sShell);
            RGB rgb = dialog.open();
            if ( rgb != null ){
                Color color = new Color( sShell.getDisplay(), rgb );
                styleRange.background=color;
            }
        }
        //如果单击设置前景色按钮, 则要设置 foreground 属性
        else if (event.widget==forColorButton){
            ColorDialog dialog = new ColorDialog(sShell);
            RGB rgb = dialog.open();
            if ( rgb != null ){
                Color color = new Color( sShell.getDisplay(), rgb );
                styleRange.foreground=color;
            }
        }
        //最后将新的样式应用于文本
        styledText.setStyleRange( styleRange );
    }
};
//为几个按钮注册选中事件
boldButton.addListener( SWT.Selection, listener);
italicButton.addListener( SWT.Selection, listener);
underlineButton.addListener( SWT.Selection, listener);
bgColorButton.addListener( SWT.Selection, listener);
forColorButton.addListener( SWT.Selection, listener);
}

```

通过这个小程序，可以总结出有关对 `StyleText` 文本选中的一些常用方法，如下：

- ❑ 获得选择文本所在的位置，返回值为 `Point` 类型：`getSelection()`。其中 `Point` 对象的 `x` 属性表示字符开始位置，`y` 选中最后一个字符的位置。
- ❑ 获得选择文本所在的位置，返回值为 `Point` 类型：`getSelectionRange()`。其中 `Point` 对象的 `x` 属性表示字符开始位置，`y` 表示选中字符的长度。
- ❑ 获得选中文本的长度，返回 `int` 值：`getSelectionCount()`。
- ❑ 获得所选择的字符串，返回 `String` 值：`getSelectionText()`。

例如，现在有这样一段文本“abcdef”选中了其中的“cde”。那么这几种方法的返回值是：

```
Point p1 = styleText.getSelection();//p1.x=2, p1.y=5
Point p2 = styleText.getSelectionRange();//p2.x=2, p2.y=3
int count = styleText.getSelectionCount();//count=3
String text = styleText.getSelectionText();//text="cde"
```

9.11.4 自动为数字字符着色

在文本改变事件中，`ExtendedModify` 事件是 `StyleText` 对象所特有的，该事件能够携带更多的信息，如属性 `replacedText` 能返回更改之前的信息。例如，将文本中的字符“AA”替换为“BB”，则 `replacedText` 的值为更改字符之前的值“AA”。

下面就以 `ExtendedModify` 为例，实现这样一个功能，当输入数字时，可以让数字显示为红色，用以区别其他字符。程序运行时如图 9.65 所示。

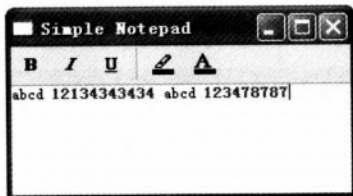


图 9.65 输入数字改变颜色

该功能实现的代码如下：

//注册 `ExtendedModify` 事件

```
styledText.addExtendedModifyListener( new ExtendedModifyListener(){
    public void modifyText(ExtendedModifyEvent event) {
        //获得修改字符的最后位置
        int end = event.start + event.length - 1;
        //如果为插入字符操作
        if (event.start <= end) {
            //获得插入的字符 StyleRange 对象
            String text = styledText.getText(event.start, end);
            //创建一个 list 对象保存所有的
            java.util.List ranges = new java.util.ArrayList();
            //循环输入每一个字符，如果有字符为数字，则将该字符的位置保存到一个
```


StyleRange 对象中

```

        for (int i = 0; i < text.length(); i++) {
            if ("0123456789".indexOf(text.charAt(i)) > -1)
                ranges.add(new StyleRange(event.start + i, 1, event.display.
getSystemColor( SWT.COLOR_RED), null, SWT.BOLD));
        }
        //如果保存 StyleRange 对象的 list 不为空, 则将 list 中的 StyleRange 应用到格
式化文本中
        if (!ranges.isEmpty())
            styledText.replaceStyleRanges(event.start, event.length, (StyleRange[])
ranges.toArray(new StyleRange[0]));
    }
});

```

该程序实现的大致原理是: 当输入字符后, 会判断该字符是否为数字, 若为数字则为该字符设置 StyleRange 对象, 使之红色显示。值得注意的是, 输入字符时可以从剪贴板上粘贴过来的字符串, 这时需要对每个字符进行判断。

9.11.5 换行自动设置背景颜色

除了上述的 ExtendedModify 事件是 StyleText 对象所独有的外, LineGetBackground 和 LineGetStyle 事件也是 StyleText 对象所独特的。它们都是在换行时触发的, 但 LineGetStyle 比 LineGetBackground 携带更多的事件触发的信息。

下面写一个小功能来看一下如何使用 LineGetBackground 事件, 该功能是, 当用户输入一行的字符串中含有“import”字符时, 就将该行的背景色设置为灰色, 效果如图 9.66 所示。

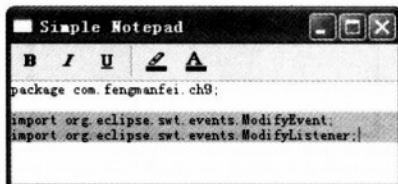


图 9.66 带有字符串“import”时显示灰色背景

该功能实现的代码很简单, 如下:

```

//注册背景色改变事件
styledText.addLineBackgroundListener(new LineBackgroundListener(){
    public void lineGetBackground(LineBackgroundEvent event) {
        //获得当前行的文本
        String text = event.lineText;
        //如果在文本中找到 import 关键字, 则设置该行的背景色为灰色
        if (text.indexOf("import") > -1)
            event.lineBackground = styledText.getDisplay().getSystemColor(SWT.COLOR_GRAY);
    }
});

```

通过本小节的学习，读者已经对 `StyleText` 控件有了一定的了解，对于编辑器来说，它的功能还是非常有限的，但理解了最基本的一些对文本处理的关键事件的处理方法，对以后学习更高级的文本编辑器是非常有意义的。

9.12 浏览器

在 SWT 中，为了支持浏览器的使用，专门提供了一个包 `org.eclipse.swt.browser` 来支持对浏览器的使用。浏览器类为该包中的 `Browser` 类，另外该包下还包括与 `Browser` 对象相关的事件处理类：

- ❑ `CloseWindowListener`：当关闭浏览器时所触发的事件。
- ❑ `LocationListener`：当浏览器转到另一个地址时触发的事件。
- ❑ `OpenWindowListener`：当打开一个浏览器窗口时触发的事件。
- ❑ `ProgressListener`：当装载一个网页时进行的事件。
- ❑ `StatusTextListener`：当浏览器状态栏改变时触发的事件。
- ❑ `TitleListener`：当浏览器标题改变时触发的事件。
- ❑ `VisibilityWindowListener`：当浏览器窗口隐藏或显示时触发的事件。

其实，在 2.4 节中，使用 `Visual Editor` 创建的第一个程序就是一个带有浏览器的小程序。浏览器运行后的图可参见图 2.20。为了使读者更清楚地理解代码，特将示例中的代码作了详细的注释。下面来具体看一下这个程序的详细代码。

SimpleSWTBrowser.java

```
package firstSWT;

import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Text;
import org.eclipse.swt.browser.Browser;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.ProgressBar;

public class SimpleSWTBrowser {

    //定义浏览器的标题
    public static final String APP_TITLE = "Simple SWT Browser";
    //定义主页的 URL
    public static final String HOME_URL = "http://www.eclipse.org/vep/";
    //声明主窗口和其他控件
    private org.eclipse.swt.widgets.Shell sShell = null;
    private Button backButton = null; //后退按钮
    private Button forwardButton = null; //前进按钮
    private Button stopButton = null; //停止按钮
```

```
private Text locationText = null;//显示 URL 的文本框
private Button goButton = null;//转向按钮
private Browser browser = null;//浏览器对象
private Button homeButton = null;//主页按钮
private Label statusText = null;//显示浏览器状态的文本框
private ProgressBar progressBar = null;//装载页面时的进度条
private Button refreshButton = null;//刷新按钮

//初始化浏览器
private void createBrowser() {
    org.eclipse.swt.layout.GridData gridData3 = new org.eclipse.swt.layout.GridData();
    //创建浏览器对象
    browser = new Browser(sShell, SWT.BORDER);
    gridData3.horizontalSpan = 7;
    gridData3.horizontalAlignment = org.eclipse.swt.layout.GridData.FILL;
    gridData3.verticalAlignment = org.eclipse.swt.layout.GridData.FILL;
    gridData3.grabExcessVerticalSpace = true;
    //设置浏览器布局
    browser.setLayoutData(gridData3);
    //为浏览器注册标题改变事件
    browser.addTitleListener(new org.eclipse.swt.browser.TitleListener() {
        public void changed(org.eclipse.swt.browser.TitleEvent e) {
            sShell.setText(APP_TITLE + " - " + e.title);
        }
    });
    //为浏览器注册地址改变事件
    browser.addLocationListener(new org.eclipse.swt.browser.LocationListener() {
        public void changing(org.eclipse.swt.browser.LocationEvent e) {
            locationText.setText(e.location);
        }

        public void changed(org.eclipse.swt.browser.LocationEvent e) {
        }
    });
    //为浏览器注册装载网页事件
    browser.addProgressListener(new org.eclipse.swt.browser.ProgressListener() {
        //当装载时，设置装载的进度，并且设置停止按钮可用
        public void changed(org.eclipse.swt.browser.ProgressEvent e) {
            if (!stopButton.isEnabled() && e.total != e.current) {
                stopButton.setEnabled(true);
            }
            progressBar.setMaximum(e.total);
            progressBar.setSelection(e.current);
        }
        //装载完成后设置停止按钮、后退按钮、前进按钮和进度条的状态
        public void completed(org.eclipse.swt.browser.ProgressEvent e) {
            stopButton.setEnabled(false);
            backButton.setEnabled(browser.isBackEnabled());
            forwardButton.setEnabled(browser.isForwardEnabled());
        }
    });
}
```



```

        progressBar.setSelection(0);
    }
});
//注册浏览器状态改变事件
browser.addStatusTextListener(new org.eclipse.swt.browser.StatusTextListener() {
    public void changed(org.eclipse.swt.browser.StatusTextEvent e) {
        statusText.setText(e.text);
    }
});
//初始状态打开主页的 URL
browser.setUrl(HOME_URL);
}

public static void main(String[] args) {
    org.eclipse.swt.widgets.Display display = org.eclipse.swt.widgets.Display.getDefault();
    SimpleSWTBrowser thisClass = new SimpleSWTBrowser();
    thisClass.createSShell();
    thisClass.sShell.open();

    while (!thisClass.sShell.isDisposed()) {
        if (!display.readAndDispatch())
            display.sleep();
    }
    display.dispose();
}

//创建窗口和窗口的控件
private void createSShell() {
    sShell = new org.eclipse.swt.widgets.Shell();
    org.eclipse.swt.layout.GridLayout gridLayout1 = new GridLayout();
    org.eclipse.swt.layout.GridData gridData2 = new org.eclipse.swt.layout.GridData();
    org.eclipse.swt.layout.GridData gridData4 = new org.eclipse.swt.layout.GridData();
    org.eclipse.swt.layout.GridData gridData5 = new org.eclipse.swt.layout.GridData();
    org.eclipse.swt.layout.GridData gridData6 = new org.eclipse.swt.layout.GridData();
    org.eclipse.swt.layout.GridData gridData7 = new org.eclipse.swt.layout.GridData();
    org.eclipse.swt.layout.GridData gridData8 = new org.eclipse.swt.layout.GridData();
    backButton = new Button(sShell, SWT.ARROW | SWT.LEFT);
    forwardButton = new Button(sShell, SWT.ARROW | SWT.RIGHT);
    stopButton = new Button(sShell, SWT.NONE);
    refreshButton = new Button(sShell, SWT.NONE);
    homeButton = new Button(sShell, SWT.NONE);
    locationText = new Text(sShell, SWT.BORDER);
    goButton = new Button(sShell, SWT.NONE);
    createBrowser();
    progressBar = new ProgressBar(sShell, SWT.BORDER);
    statusText = new Label(sShell, SWT.NONE);
    sShell.setText(APP_TITLE);
    sShell.setLayout(gridLayout1);
    gridLayout1.numColumns = 7;
}

```

```
backButton.setEnabled(false);
backButton.setToolTipText("Navigate back to the previous page");
backButton.setLayoutData(gridData6);
forwardButton.setEnabled(false);
forwardButton.setToolTipText("Navigate forward to the next page");
forwardButton.setLayoutData(gridData5);
stopButton.setText("Stop");
stopButton.setEnabled(false);
stopButton.setToolTipText("Stop the loading of the current page");
goButton.setText("Go!");
goButton.setLayoutData(gridData8);
goButton.setToolTipText("Navigate to the selected web address");
gridData2.grabExcessHorizontalSpace = true;
gridData2.horizontalAlignment = org.eclipse.swt.layout.GridData.FILL;
gridData2.verticalAlignment = org.eclipse.swt.layout.GridData.CENTER;
locationText.setLayoutData(gridData2);
locationText.setText(HOME_URL);
locationText.setToolTipText("Enter a web address");
homeButton.setText("Home");
homeButton.setToolTipText("Return to home page");
statusText.setText("Done");
statusText.setLayoutData(gridData7);
gridData4.horizontalSpan = 5;
progressBar.setLayoutData(gridData4);
progressBar.setEnabled(false);
progressBar.setSelection(0);
gridData5.horizontalAlignment = org.eclipse.swt.layout.GridData.FILL;
gridData5.verticalAlignment = org.eclipse.swt.layout.GridData.FILL;
gridData6.horizontalAlignment = org.eclipse.swt.layout.GridData.FILL;
gridData6.verticalAlignment = org.eclipse.swt.layout.GridData.FILL;
gridData7.horizontalSpan = 1;
gridData7.grabExcessHorizontalSpace = true;
gridData7.horizontalAlignment = org.eclipse.swt.layout.GridData.FILL;
gridData7.verticalAlignment = org.eclipse.swt.layout.GridData.CENTER;
gridData8.horizontalAlignment = org.eclipse.swt.layout.GridData.END;
gridData8.verticalAlignment = org.eclipse.swt.layout.GridData.CENTER;
refreshButton.setText("Refresh");
refreshButton.setToolTipText("Refresh the current page");
sShell.setSize(new org.eclipse.swt.graphics.Point(553, 367));
//注册显示地址的文本框事件
locationText.addMouseListener(new org.eclipse.swt.events.MouseAdapter() {
    public void mouseUp(org.eclipse.swt.events.MouseEvent e) {
        locationText.selectAll();
    }
});
locationText.addKeyListener(new org.eclipse.swt.events.KeyAdapter() {
    public void keyPressed(org.eclipse.swt.events.KeyEvent e) {
        // Handle the press of the Enter key in the locationText.
        // This will browse to the entered text.
    }
});
```

```
        if (e.character == SWT.LF || e.character == SWT.CR) {
            e.doit = false;
            browser.setUrl(locationText.getText());
        }
    }
});
refreshButton.addSelectionListener(new org.eclipse.swt.events.SelectionAdapter() {
    public void widgetSelected(org.eclipse.swt.events.SelectionEvent e) {
        browser.refresh();//重新载入
    }
});
locationText.addSelectionListener(new org.eclipse.swt.events.SelectionAdapter() {
    public void widgetSelected(org.eclipse.swt.events.SelectionEvent e) {
        browser.setUrl(locationText.getText());//设置浏览器的指向的 url
    }
});
stopButton.addSelectionListener(new org.eclipse.swt.events.SelectionAdapter() {
    public void widgetSelected(org.eclipse.swt.events.SelectionEvent e) {
        browser.stop();//停止装载网页
    }
});
backButton.addSelectionListener(new org.eclipse.swt.events.SelectionAdapter() {
    public void widgetSelected(org.eclipse.swt.events.SelectionEvent e) {
        browser.back();//后退
    }
});
forwardButton.addSelectionListener(new org.eclipse.swt.events.SelectionAdapter() {
    public void widgetSelected(org.eclipse.swt.events.SelectionEvent e) {
        browser.forward();//前进
    }
});
homeButton.addSelectionListener(new org.eclipse.swt.events.SelectionAdapter() {
    public void widgetSelected(org.eclipse.swt.events.SelectionEvent e) {
        browser.setUrl(HOME_URL);//设置主页
    }
});
goButton.addSelectionListener(new org.eclipse.swt.events.SelectionAdapter() {
    public void widgetSelected(org.eclipse.swt.events.SelectionEvent e) {
        browser.setUrl(locationText.getText());//转向地址的网页
    }
});
}
```

以上的程序代码虽然很长,但关键是学会如何使用 Browser 对象和一些方法,并结合相应的事件来改变各控件的状态,请读者仔细阅读代码部分。

9.13 本章小结

通过本章的学习，读者应该对 SWT 的各种控件有了一个全面的了解。但是，作为实际的项目开发中，很少用到这些控件，尤其是在开发 Eclipse 插件或是 RCP 程序中，通常是扩展 Eclipse 中的类。而 JFace 中也提供了对这些控件的扩展，使之功能更加强大。



第 10 章 SWT 中的拖放支持

本章将详细讲述 SWT 中对拖放 (Drag and Drop) 的支持, 首先以一个拖放的示例程序使读者对拖放的过程有一个大致的了解; 然后深入到拖放的理论, 包括什么是拖放源、拖放目标和传输数据等; 最后, 讲述与传输数据密切相关的一些对剪贴板的操作。

10.1 可拖放的树

在应用程序中, 通常为了简化操作使用拖放功能。举一个很小的例子, 在使用 Word 编辑文字时, 通常可以选中几个字符, 然后按住鼠标左键将选中的字符拖动到文档的某一个位置, 释放后, 选中的文字就移动到了新的位置上, 这个拖放的操作过程相当于剪切和粘贴的作用。

当然, 这只是简单地拖放文字, 还可以拖放更为复杂的对象。例如, 在 Eclipse 的工作区中, 可以通过拖放的方式改变视图的布局。

第 9 章讲述树是不能够拖动的, 当定义了拖放对象后, 可以将普通的树变成可拖放的树。如图 10.1 所示为一个可拖动的树示意图。

实现该拖动树的代码如下:

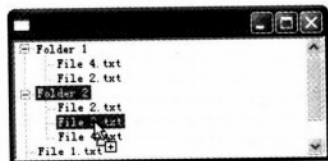


图 10.1 可拖动的树

DragTree.java

```
package com.fengmanfei.ch10;
import java.io.File;
import org.eclipse.swt.SWT;
import org.eclipse.swt.dnd.*;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.widgets.*;

public class DragTree {

    public static void main(String[] args) {
        Display display = new Display();
        Shell shell = new Shell(display);
        shell.setLayout(new FillLayout());
        //创建一个树对象
        final Tree tree = createFolderTree(shell);
        //定义拖放源对象
```



```

DragSource dragSource = new DragSource(tree, DND.DROP_MOVE|DND.DROP_
COPY);
//设置传输的数据为文本型 String 类型
dragSource.setTransfer(new Transfer[] { TextTransfer.getInstance()});
//注册拖放源时的事件处理
dragSource.addDragListener(new DragSourceListener() {
    public void dragStart(DragSourceEvent event) {
        if (tree.getSelectionCount() == 0)
            event.doit = false;
    }
    public void dragSetData(DragSourceEvent event) {
        if (TextTransfer.getInstance().isSupportedType(event.dataType)) {
            event.data = tree.getSelection()[0].getText(0);
        }
    }
    public void dragFinished(DragSourceEvent event) {
    }
});
//定义拖放目标对象
DropTarget dropTarget = new DropTarget(tree, DND.DROP_MOVE | DND.DROP_
DEFAULT|DND.DROP_COPY);
//设置目标对象可传输的数据类型
dropTarget.setTransfer(new Transfer[] { TextTransfer.getInstance()});
//注册目标对象的事件处理
dropTarget.addDropListener(new DropTargetListener() {
    public void dragEnter(DropTargetEvent event) {
        if (event.detail == DND.DROP_DEFAULT)
            event.detail = DND.DROP_COPY;
    }
    public void dragOperationChanged(DropTargetEvent event) {
        if (event.detail == DND.DROP_DEFAULT)
            event.detail = DND.DROP_COPY;
    }
    public void dragOver(DropTargetEvent event) {
        event.feedback = DND.FEEDBACK_EXPAND | DND.FEEDBACK_SELECT;
    }
    //当松开鼠标时触发的事件
    public void drop(DropTargetEvent event) {
        if (event.item == null)
            return;
        //首先获得目标对象中拖动的树节点
        TreeItem eventItem = (TreeItem) event.item;
        if (TextTransfer.getInstance().isSupportedType(event.currentDataType)) {
            //获得数据源设置的字符串
            String s = (String) event.data;
            //在当前位置插入一个节点
            TreeItem newItem = null ;
            if (eventItem.getParentItem()==null)
                newItem = new TreeItem(eventItem.getParent(),SWT.NONE);

```



```

        else
            newItem = new TreeItem(eventItem.getParentItem(), SWT.NONE);
            newItem.setText(s);
        }
    }
    public void dragLeave(DropTargetEvent event) {
    }
    public void dropAccept(DropTargetEvent event) {
    }
});
shell.setSize(300, 150);
shell.open();
while (!shell.isDisposed()) {
    if (!display.readAndDispatch())
        display.sleep();
}
display.dispose();
}
//创建树，利用目录结构来初始化树数据
public static Tree createFolderTree(Composite parent) {
    Tree tree = new Tree(parent, SWT.BORDER | SWT.SINGLE);
    File root = new File("F:\\Temp");
    File[] files = root.listFiles();
    for (int i = 0; i < files.length; i++) {
        TreeItem item = new TreeItem(tree, SWT.NONE);
        item.setText(files[i].getName());
        if (files[i].isDirectory())
            setDirectory(files[i], item);
    }
    return tree;
}
//设置子树的子孙数据，递归调用该方法
public static void setDirectory(File file, TreeItem parent) {
    File[] files = file.listFiles();
    for (int i = 0; i < files.length; i++) {
        TreeItem item = new TreeItem(parent, SWT.NONE);
        item.setText(files[i].getName());
        if (files[i].isDirectory())
            setDirectory(files[i], item);
    }
}
}
}

```

该程序的代码虽然比较复杂，但弄清楚拖放源、拖放目标和传输数据之间的关系，就不难理解其中的道理了，下面就来具体看一下实现拖放的详细过程。

10.2 拖放原理概述

SWT 中有关拖放操作的类都在 `org.eclipse.swt.dnd` 包中。该包中主要的接口和类如表 10.1 所示。

表 10.1 `org.eclipse.swt.dnd` 的主要类说明

类 名	功 能 描 述
Clipboard	剪贴板类，可以实现对剪贴板的操作
DND	类似于 SWT 类，封装了拖放时所用的一些常量
DragSource	拖放源类，可将一个控件定义为拖放源
DropTarget	拖放目标类，可将一个空间定义为拖放目标
DragSourceListener 和 DragSourceAdapter	拖放源控件时所触发的事件，触发事件的具体信息通过 <code>DragSourceEvent</code> 传递过来
DropTargetListener 和 DropTargetAdapter	拖放目标控件时所触发的事件，触发事件的具体信息通过 <code>DropTargetEvent</code> 传递过来
Transfer	数据传输类，用于将 Java 数据转化为系统本地的数据，反之亦然。通常用于拖放操作和剪贴板操作。通常使用其子类 <code>TextTransfer</code> 类、 <code>FileTransfer</code> 类和 <code>RTFTransfer</code> 类等

看似简单的拖放操作，其实过程相当复杂，现在就将一个拖放过程分解开来，仔细分析一下拖放的过程。

(1) 要有一个拖放源，也就是“拖的是什么”。简单的理解就是可以进行拖放的控件。只有光标进入到可拖放控件的区域，才可以开始拖的操作。SWT 中使用 `DragSource` 类来定义一个拖放源对象。

(2) 还要有一个拖放目标，也就是“放到哪里”。简单的理解就是可接受源的目标控件。只有当光标进入到目标控件的区域时，才可以进行放的操作。SWT 中使用 `DragTarget` 类来定义一个拖放目标对象。

(3) 要在拖和放操作之间搭建一个桥梁，将拖的数据放到目标对象中。

拖放操作无非是将源的数据放置到目标对象中，SWT 中的处理是在拖的过程中将 Java 的数据转化为本地保存的全局变量数据，然后在放的过程中，再从本地保存的全局变量中将数据取出。SWT 中使用 `Transfer` 类来定义一个转化数据的过程。

如图 10.2 所示为整个拖放过程的示意图。

从图 10.2 中可以看出，要想理解拖放的全过程，关键是要理解 3 个概念：拖放源（`DragSource`）、传输数据（`Transfer`）和拖放目标（`DragTarget`）。在以下的几个小节中就来看详细一下这 3 个类。

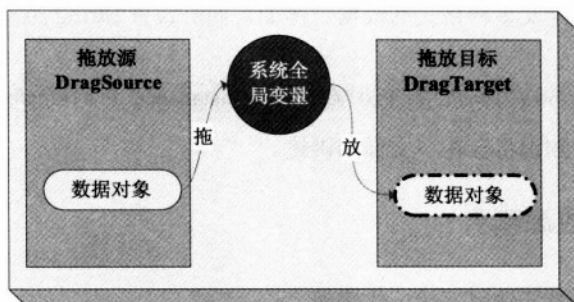


图 10.2 拖放过程示意图

10.3 拖放源 (DragSource)

10.3.1 创建拖放源对象

拖放源是定义了可拖动的控件。例如，以下所示将一个表格定义为拖放源对象。

```
Table table = new Table(shell, SWT.NONE);
//...省略中间代码
DragSource source = new DragSource(table, DND.DROP_MOVE | DND.DROP_COPY);
```

📌 注意：一个控件只能绑定一个拖放源。

创建拖放源时，第一个参数表示的是该拖放源绑定的控件对象，第二个参数表示拖放过程中可支持的操作类型。操作类型决定了在目标对象 (DragTarget) 中数据传输中支持的类型。可选的常量有 DND.DROP_MOVE、DND.DROP_COPY、DND.DROP_LINK 和 DND.DROP_NONE。例如，这里示例中的拖放源可支持的类型为可移动 (DND.DROP_MOVE) 或复制 (DND.DROP_COPY)。

📌 注意：有关拖放类中涉及的常量是在 org.eclipse.swt.dnd.DND 类中，而不是之前使用的 org.eclipse.swt.SWT 中的常量。

10.3.2 定义拖放源数据传输类型

定义完拖放源对象后便要定义传输数据类型，也就是在拖放过程中将什么样格式的数据放置到本地系统的全局变量中。例如，将 Java 提供的 String 类型的数据存放到全局变量中，使用的是 TextTransfer 对象，代码如下所示：

```
source.setTransfer(new Transfer[] {TextTransfer.getInstance()});
```

若要设置 String[] 数组类型的数据格式，代码如下：

```
source.setTransfer(new Transfer[] {FileTransfer.getInstance()});
```

当然也可用同时定义多种格式的数据。例如，同时设置 `String` 类型和 `String[]` 类型的格式数据，代码如下：

```
source.setTransfer(new Transfer[] { TextTransfer.getInstance(), FileTransfer.getInstance()});
```

有关传输数据的知识将会在下文详细讲述。

10.3.3 处理拖放源事件

最后要添加拖放事件处理方法。代码如下：


```
source.addDragListener(new DragSourceListener() {  
    // 当开始拖动源控件时  
    public void dragStart(DragSourceEvent event) {}  
    // 设置拖放过程中传输的数据  
    public void dragSetData(DragSourceEvent event) {}  
    // 当拖动完成后，释放光标所触发的事件  
    public void dragFinished(DragSourceEvent event) {}  
});
```

当开始拖动源对象时，要触发以下 3 个事件：

1. `dragStart()`：开始拖动处触发的事件

在光标进入到拖放源区域，并且此时按中光标开始移动的一瞬间所触发的事件。该事件一般用于判断是否开始拖动。如果不可以拖动，则将 `event` 对象的 `doit` 属性设置为 `false`。例如，在一个表格作为拖放源的程序中，要判断该表格有选中的行时才能开始拖动，实现该功能的代码如下所示：

```
public void dragStart(DragSourceEvent event) {  
    if (table.getSelectionCount() == 0)  
        event.doit = false;  
}
```

 注意：`dragStart` 方法调用后，此时没有设置 `Transfer` 对象，则会取消拖动操作。

2. `dragSetData()`：设置数据事件

该方法只有在拖放到目标对象区域中，并且在 `dragFinished()` 事件触发之前会多次调用，作用是设置放置在全局变量中的数据，该方法非常重要。与目标事件处理的接口，通常每次调用该方法时要判断 `event.dataType` 类型，然后根据是否支持的数据传输类型来设置数据，并通过 `event.data` 属性来设置传输的数据类型。


例如，在一个表格作为拖放源的程序中，将表格中选择的一行文本（`String`）复制到全局变量中，并且将表格中一行字符串数组（`String[]`）复制到全局变量中。代码如下：

```
public void dragSetData(DragSourceEvent event) {  
    TableItem item = table.getSelection()[0];  
    if (TextTransfer.getInstance().isSupportedType(event.dataType)){  
        //复制到全局变量的数据为 String 型
```

```

        event.data = item.getText(0) + ", " + item.getText(1);
    }
    if (FileTransfer.getInstance().isSupportedType(event.dataType)) {
        //复制到全局变量的数据为 String[]数组类型
        event.data =new String[]{" item.getText(0)", " item.getText(1)"};
    }
}

```

 注意：如果未将光标移动到目标区域释放，将不会触发 dragSetData()事件。

3. dragFinished(): 完成拖放后的事件

拖放完成后，释放鼠标，此时触发该事件。当释放鼠标时，此时未在目标对象的区域或者在拖放的过程中按 Esc 键，则此时 event.doit 属性值为 false 并且 event.detail 属性值为 DND.DROP_NONE。当鼠标释放时，此时在目标对象的区域，event.doit 属性值为 true，而 event.detail 属性值要由目标对象中的设置的操作而定，此时可以通过 event.detail 属性来判断在目标对象中所进行的操作。表 10.2 显示了可在目标对象中设置的 event.detail 常量。

表 10.2 event.detail 常量

操作样式常量	说 明
DND.DROP_COPY	目标对象复制了数据
DND.DROP_MOVE	目标对象剪切了数据，需要删除原来的数据
DND.DROP_LINK	目标对象创建了快捷方式，一般使用于对文件的操作
DND.DROP_TARGET_MOVE	目标对象改变了数据的位置，通常是用于移动文件

依据 event.detail 属性，通常可在 dragFinished()事件中处理一些拖放完成后的善后工作，比如说移动表格一行后，将移动的这一行在原表格删除等。若实现这样一个功能，代码如下：

```

public void dragFinished(DragSourceEvent event) {
    //如果目标对象的处理事件中设置的 event.detail 为 DND.DROP_MOVE
    if ( event.detail==DND.DROP_MOVE){
        TableItem item = table.getSelection()[0];
        int index = table.indexOf(item);
        if (index>-1)
            table.remove(index);
    }
}

```

综上所述，当拖放源对象时，所触发的事件的顺序可以是以下几种：





- ☐ dragStart。例如没有为源对象设置 Transfer 对象。
- ☐ dragStart、多次 dragSetData、dragFinished。例如一个完整的拖放过程。
- ☐ dragStart、dragFinished。在拖放的过程中按 Esc 键，取消了拖放操作。

10.4 拖放目标 (DragTarget)

10.4.1 定义目标对象

如果说拖放源 (DragSource) 是数据传输过程中的数据提供者, 那么拖放目标 (DropTarget) 就是数据接收者。当鼠标进入到目标区域时, 可以通过鼠标的变化来判断是否进入到了有效的目标区域中。在 Windows 操作系统下, 拖动过程中鼠标的图标如表 10.3 所示。

表 10.3 拖动过程的鼠标样式 (Windows)

样式常量	鼠标样式
DND.DROP_MOVE	
DND.DROP_COPY	
DND.DROP_LINK	
DND.DROP_NONE	

定义拖放目标对象的控件可以是拖放源本身也可以是除了拖放源的其他控件。

❑ 例如, 在 10.3.3 节的例子中, 将拖放源本身的表格也作为拖放目标的代码如下:

```
Table table = new Table(shell, SWT.NONE);
//...省略中间代码
DragSource source = new DragSource(table, DND.DROP_MOVE | DND.DROP_COPY);
//...省略中间代码
DropTarget target = new DropTarget(table, DND.DROP_DEFAULT | DND.DROP_MOVE);
```

❑ 例如, 拖放目标定义为另一个表格的代码如下:

```
Table table = new Table(shell, SWT.NONE);
Table targetTable = new Table(shell, SWT.NONE);
//...省略中间代码
DragSource source = new DragSource(table, DND.DROP_MOVE | DND.DROP_COPY);
//...省略中间代码
DropTarget target = new DropTarget(targetTable, DND.DROP_DEFAULT | DND.DROP_MOVE);
```

与创建拖放源一样, 创建拖放目标对象的第一个参数为所绑定的控件, 第二个参数为目标对象所支持的操作类型。与之不同的是, 如果指定了 DND.DROP_DEFAULT, 则会影响事件处理的方法 dragEnter 事件和 dragOperationChanged 事件。有关它的详细讲述请参阅 10.4.3 节有关目标源事件处理的方法部分。

⚠注意: 本小节所讲的示例都是源和目标不是同一个控件, 对源和目标为同一个控件的示例程序请参阅 10.6 节。

10.4.2 定义目标对象的数据传输类型

与定义拖放源数据传输类型一样，对拖放目标对象也要定义接收数据类型。代码如下：

```
TextTransfer textTransfer = TextTransfer.getInstance();
FileTransfer fileTransfer = FileTransfer.getInstance();
Transfer[] types = new Transfer[] {fileTransfer, textTransfer};
target.setTransfer(types);
```

10.4.3 处理拖放目标事件

与拖放源对象的处理事件类似，当光标进入到目标对象区域时便会触发目标对象的相关事件。为拖放目标对象注册事件的代码如下：

```
target.addDropListener(new DropTargetListener() {
    //当鼠标进入到目标对象区域的一瞬间所触发的事件
    public void dragEnter(DropTargetEvent event) {};
    //当鼠标在目标对象的区域移动时触发的事件，在一次拖放中可多次调用
    public void dragOver(DropTargetEvent event) {};
    //当鼠标离开目标对象的区域时或取消的拖动过程
    public void dragLeave(DropTargetEvent event) {};
    //改变操作时触发的事件，一般是在拖动的过程中使用了辅助键（Ctrl 键和 Shift 键等）
    public void dragOperationChanged(DropTargetEvent event) {};
    //在完成拖放之前事件提供了最后一次定义数据类型的机会
    public void dropAccept(DropTargetEvent event) {}
    //释放鼠标，完成拖放所触发的事件
    public void drop(DropTargetEvent event) {}
});
```

下面就来对每个事件详细地分析一下。

1. dragEnter

光标进入到目标对象的区域时触发的事件。如果在定义目标对象时使用了 DND.DROP_DEFAULT 样式，则当触发此事件时，同时也没有按下辅助键，event.detail 属性值则为 DND.DROP_DEFAULT。

如果在定义目标对象时没有使用 DND.DROP_DEFAULT 样式，则当触发此事件时，同时也没有按下辅助键，event.detail 属性值则为 DND.DROP_MOVE。event.detail 属性值决定了显示的光标状态。

例如，以下代码显示了在进入时，如果未按其他的辅助键将鼠标改成复制的样式，前提条件是创建目标对象的样式中使用了 DND.DROP_DEFAULT 样式。

```
public void dragEnter(DropTargetEvent event) {
    if (event.detail == DND.DROP_DEFAULT)
        event.detail = DND.DROP_COPY; //改变鼠标为复制样式
}
```

2. dragOver

光标进入目标对象区域中的事件。光标进入目标对象区域中会被重复不停地触发该事件，如果光标此时是静止的，该方法仍就会有规则地按一定时间间隔被触发。通常在表格（Table）和树（Tree）拖动时，用于处理当前移动的表格或树的状态。

例如，在目标对象为表格时，可以使用户看到表格的状态为选中一行，并且当表格数据很多时，可以滚动到可见的行上。

```
public void dragOver(DropTargetEvent event) {
    event.feedback = DND.FEEDBACK_SELECT | DND.FEEDBACK_SCROLL;
}
```

event.feedback 属性可以设置拖放过程中，目标对象的状态，该属性仅对表格和树有效。该属性可选的参数及代表的意义如表 10.4 所示。

表 10.4 event.feedback 属性值

属 性 值	描 述
DND.FEEDBACK_SELECT	光标当前的行或树节点处于选中状态，仅适用于树（Tree）和表格（Table）
DND.FEEDBACK_SCROLL	可以放到未显示的行或树节点，仅适用于树（Tree）和表格（Table）
DND.FEEDBACK_EXPAND	可展开当前光标所在的节点，仅适用于树（Tree）
DND.FEEDBACK_INSERT_BEFORE	在当前光标前插入标记，仅适用于树（Tree）和表格（Table）
DND.FEEDBACK_INSERT_AFTER	在当前光标后插入标记，仅适用于树（Tree）和表格（Table）
DND.FEEDBACK_NONE	不显示任何拖放效果

3. dragOperationChanged

用户按下或放开辅助键时触发的事件。当用户按下或放开辅助按键时，例如 Ctrl、Shift 等键时，则触发该事件。此时如果没有按下辅助键，event.detail 属性值则为 DND.DROP_DEFAULT。例如，当没有按下辅助键时，将光标的样式设置为复制的样式，代码如下：

```
public void dragOperationChanged(DropTargetEvent event) {
    if (event.detail == DND.DROP_DEFAULT)
        event.detail = DND.DROP_COPY;
}
```

4. dragLeave

当鼠标离开目标对象时触发的事件。当鼠标离开目标对象时，或在拖放过程中按下了 Esc 键，也会触发该事件，一般可用于在完成拖放后释放一些资源。

5. dropAccept

在完成拖放之前事件提供了最后一次定义数据类型的机会。

6. drop

拖放完成后触发的事件。当用户在有效的目标区域内释放鼠标，并且此时获得了有效的数据类型 currentDataType，此时可以触发该事件。通过 event.data 可以获得由拖放源存放到的全局变量中的数据。例如，此时将在拖放源事件 dragSetData() 方法中设置的数据取出来的

代码如下：

```
public void drop(DropTargetEvent event) {
    TableItem item = (TableItem)event.item;
    if (item == null) {
        event.detail = DND.DROP_NONE;
        return;
    }
    if (fileTransfer.isSupportedType(event.currentDataType)) {
        String[] files = (String[])event.data;
        if (files != null && files.length > 0) {
            item.setText(files[0]);
        }
    }
    if (textTransfer.isSupportedType(event.currentDataType)) {
        String text = (String)event.data;
        if (text != null) {
            item.setText(text);
        }
    }
}
```

综上所述，进入到目标对象时，所触发的事件的顺序可以是以下几种：

- ☐ dragEnter, dragLeave。
- ☐ dragEnter, 多次 dragOver, dragLeave。
- ☐ dragEnter, 多次 dragOver, dragLeave, dropAccept。
- ☐ dragEnter, 多次 dragOver, dragLeave, dropAccept, drop。

10.5 传输数据（Transfer）

传输数据类（Transfer）提供了将 Java 数据和本地操作系统数据交换的手段，该类是一个抽象类，在使用传输数据类时，通常使用 Transfer 子类。例如，String 类型的数据需要使用 TextTransfer 类，String[] 数组类型的数据要使用 FileTransfer 类。SWT 中 Transfer 与其子类的继承关系图如图 10.3 所示。

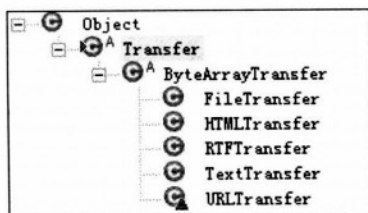


图 10.3 数据传输类的继承关系图

如果当前提供的传输数据类型不能满足需要时，可以自定义数据传输类，需要继承

ByteArrayTransfer 类, 并且覆盖其 javaToNative (Object object, TransferData transferData)方法和 nativeToJava(TransferData transferData)方法。该部分知识比较复杂, 读者如果有兴趣, 可以参阅 API 文档中的 ByteArrayTransfer 类的参考, 其中有详细的说明。

SWT 中提供的几种数据传输类以及它们分别支持的 Java 数据类型, 如表 10.5 所示。

表 10.5 数据传输类及其说明

类 名	Java 数据类型	举 例
TextTransfer	String	"Hello World"
FileTransfer	String[]	File file1 = new File("C:\temp\file1"); File file2 = new File("C:\temp\file2"); new String[] {file1.getAbsolutePath(), file2.getAbsolutePath()}
HTMLTransfer	String	"This is a paragraph of text."
RTFTransfer	String	"{\rtf1 {\colortbl;\red255\green0\blue0;}\uc1\b\i Hello World}"

注意: 在 JFace 中, 有一些为了拖动 Eclipse 平台使用的数据传输类, 如 PluginTransfer、ResourceTransfer 类等。

10.6 综合示例：简单购物车

最后看一个综合性的小程序, 如何实现拖放的功能。该程序的功能是: 左侧有一个产品列表, 显示了不同的产品以及价格, 右侧是一个购物车列表, 显示了用户已选的产品和数量。当用户将所要购买的产品拖动到购物车列表时, 产品会自动添加到购物车列表中, 并自动计算数量和总价格。如图 10.4 所示为程序运行后的效果图。

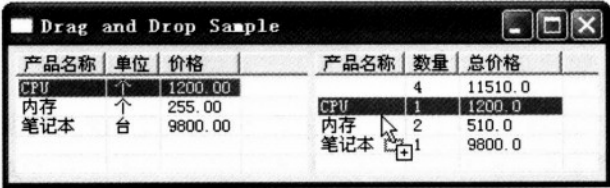


图 10.4 拖放程序示意图

该拖放程序实现的代码如下:

DragDropSample.java

```
package com.fengmanfei.ch10;

import org.eclipse.swt.SWT;
import org.eclipse.swt.dnd.*;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.widgets.*;

public class DragDropSample {
```

```

public static void main(String[] args) {
    Display display = new Display();
    Shell shell = new Shell(display);
    shell.setText("Drag and Drop Sample");
    GridLayout layout = new GridLayout();
    layout.numColumns = 2;
    shell.setLayout(layout);
    // 创建左侧的产品表格
    final Table productList = createProductTable(shell);
    // 创建右侧的产品表格
    final Table shoppingCart = createCartTable(shell);
    // 创建拖放源
    DragSource source = new DragSource(productList, DND.DROP_MOVE | DND.DROP_
COPY);
    // 设置传输的数据为文本型 String 类型
    source.setTransfer(new Transfer[] { TextTransfer.getInstance() });
    // 注册当拖动源时触发的事件
    source.addDragListener(new DragSourceListener() {
        // 当开始拖动源控件时
        public void dragStart(DragSourceEvent event) {
            if (productList.getSelectionCount() == 0)
                event.doit = false;
        }
        // 设置拖放过程中传输的数据
        public void dragSetData(DragSourceEvent event) {
            if (TextTransfer.getInstance().isSupportedType(event.dataType)) {
                // 获得当前选中的一行
                TableItem item = productList.getSelection()[0];
                // 将产品和产品的单价表示成字符串的形式，用逗号分隔，例如"CPU,
1200"，并赋值给 event.data
                event.data = item.getText(0) + "," + item.getText(2);
            }
        }
        // 当拖动完成后，释放光标所触发的事件
        public void dragFinished(DragSourceEvent event) {
        }
    });

    // 定义拖放的目标控件
    DropTarget target = new DropTarget(shoppingCart, DND.DROP_COPY | DND.DROP_
DEFAULT);
    // 设置目标控件所接收的传输数据
    target.setTransfer(new Transfer[] { TextTransfer.getInstance() });
    target.addDropListener(new DropTargetListener() {
        // 当光标进入到目标控件的区域时
        public void dragEnter(DropTargetEvent event) {
            if (event.detail == DND.DROP_DEFAULT)
                event.detail = DND.DROP_COPY;
        }
    });
}

```



```

    }
    //当光标离开目标控件的区域时
    public void dragLeave(DropTargetEvent event) {
    }
    //当改变 event.detail 类型时触发的事件
    public void dragOperationChanged(DropTargetEvent event) {
        if (event.detail == DND.DROP_DEFAULT)
            event.detail = DND.DROP_COPY;
    }
    //当光标在目标区域上时
    public void dragOver(DropTargetEvent event) {
    }
    //拖动完成后触发的事件
    public void drop(DropTargetEvent event) {
        //首先获得当前目标表格项
        TableItem item = (TableItem) event.item;
        if (item == null) {
            event.detail = DND.DROP_NONE;
            return;
        }
        //如果支持文本型传输数据
        if (TextTransfer.getInstance().isSupportedType(event.currentDataType)) {
            //首先获得源数据中的字符串数据, 该数据是通过 dragSetData()方法设置的
            //例如此时获得的数据 dataInfo 为"CPU, 1200";
            String dataInfo = (String) event.data;
            if (dataInfo == null)
                return;
            //首先解析逗号分割的字符串, 逗号之前的字符为产品名, 逗号之后的字
            int index = dataInfo.indexOf(",");
            String name = dataInfo.substring(0, index);
            double price = 0.00d;
            try {
                price = Double.parseDouble(dataInfo.substring(index + 1));
            } catch (Exception e) {
                price = 0.00d;
            }
            //获得目标表格
            Table parent = item.getParent();
            TableItem it = null;
            //循环目标表格中的每一项, 是否购物车中已经添加了该产品
            for (int i = 0; i < parent.getItemCount(); i++) {
                TableItem temp = parent.getItem(i);
                if (temp.getText(0).equals(name)) {
                    it = temp;
                    break;
                }
            }
            //如果没找到, 则新建一行, 数量加 1, 并显示总价格

```

符为价格


```

        if (it == null) {
            it = new TableItem(parent, SWT.NONE);
            it.setText(0, name);
            it.setText(1, "1");
            it.setText(2, "" + price);
        } else { //如果找到, 更新该产品的数量和总价格
            int number = Integer.parseInt(it.getText(1)) + 1;
            double total = number * price;
            it.setText(1, "" + number);
            it.setText(2, "" + total);
        }
        //最后更新购物车的总价和数量
        TableItem sumItem = parent.getItem(0);
        sumItem.setText(1, Integer.parseInt(sumItem.getText(1)) + 1 + "");
        sumItem.setText(2, Double.parseDouble(sumItem.getText(2)) + price + "");
    }

    public void dropAccept(DropTargetEvent event) {
    }

});
shell.pack();
shell.open();
while (!shell.isDisposed()) {
    if (!display.readAndDispatch())
        display.sleep();
}
display.dispose();
}

//创建产品的表格
public static Table createProductTable(Composite parent) {
    Table table = new Table(parent, SWT.FULL_SELECTION | SWT.SINGLE);
    String[] heads = { "产品名称", "单位", "价格" };
    for (int i = 0; i < heads.length; i++) {
        TableColumn col = new TableColumn(table, SWT.NONE);
        col.setText(heads[i]);
    }
    TableItem item = new TableItem(table, SWT.NONE);
    item.setText(new String[] { "CPU", "个", "1200.00" });
    item = new TableItem(table, SWT.NONE);
    item.setText(new String[] { "内存", "个", "255.00" });
    item = new TableItem(table, SWT.NONE);
    item.setText(new String[] { "笔记本", "台", "9800.00" });
    table.setLayoutData(new GridData(SWT.FILL, SWT.FILL, true, true));
    table.setHeaderVisible(true);
    table.setLinesVisible(true);
    for (int i = 0; i < heads.length; i++) {
        table.getColumns()[i].pack();
    }
    return table;
}

```

```
}  
//创建购物车表格  
public static Table createCartTable(Composite parent) {  
    Table table = new Table(parent, SWT.FULL_SELECTION);  
    String[] heads = { "产品名称", "数量", "总价格" };  
    for (int i = 0; i < heads.length; i++) {  
        TableColumn col = new TableColumn(table, SWT.NONE);  
        col.setText(heads[i]);  
    }  
    TableItem item = new TableItem(table, SWT.NONE);  
    item.setText(new String[] { " ", "0", "0.00" });  
    table.setHeaderVisible(true);  
    table.setLinesVisible(true);  
    table.setLayoutData(new GridData(SWT.FILL, SWT.FILL, true, true));  
    for (int i = 0; i < heads.length; i++) {  
        table.getColumns()[i].pack();  
    }  
    return table;  
}
```

该程序实现的关键方法是在 `drop` 方法中的处理，在将产品放到购物车中时，先要检查购物车中是否已经有了该产品，如果没有，则添加一个；若有，则将该产品的数量加一，最后再更新总价值的值。

10.7 对剪贴板的操作

剪贴板读者应该非常熟悉了，SWT 提供了可以直接访问系统剪贴板的类 `Clipboard`。通过该类，可以轻松地实现对剪贴板的操作，下面以一个简单的程序为例，来讲述一下剪贴板的用法。

该程序有一个文本框，选中字符后右击，在弹出的快捷菜单中可以选择复制到剪贴板上，并且可以将剪贴板的字符粘贴到文本框中。程序运行后的效果如图 10.5 所示。

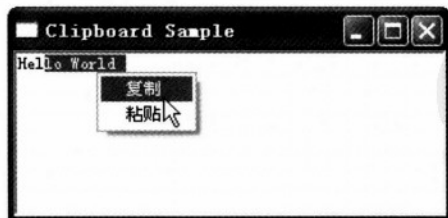


图 10.5 剪贴程序效果图

该程序实现的代码如下：

ClipboardSample.java

```
package com.fengmanfei.ch10;

import org.eclipse.swt.SWT;
import org.eclipse.swt.dnd.*;
import org.eclipse.swt.events.*;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.widgets.*;

public class ClipboardSample {
    public static void main(String[] args) {
        Display display = new Display();
        //定义剪贴板对象
        final Clipboard cb = new Clipboard(display);
        Shell shell = new Shell(display);
        shell.setLayout(new FillLayout());
        shell.setText("Clipboard Sample");
        //定义弹出式菜单
        final Text content = new Text(shell, SWT.BORDER | SWT.MULTI | SWT.WRAP);
        Menu menu = new Menu(shell, SWT.POP_UP);
        //复制菜单项
        final MenuItem copyItem = new MenuItem(menu, SWT.PUSH);
        copyItem.setText("复制");
        //注册复制菜单项事件
        copyItem.addSelectionListener(new SelectionAdapter() {
            //当选中该菜单项时
            public void widgetSelected(SelectionEvent e) {
                //如果此时未选中任何文字, 则不执行复制操作
                String selection = content.getSelectionText();
                if (selection.length() == 0)
                    return;
                //定义放入到剪贴板中的数据类型, 可同时设置多种类型, 这里指设置 String 类型
                Object[] data = new Object[] { selection };
                Transfer[] types = new Transfer[] { TextTransfer.getInstance() };
                //将字符串放入到剪贴板上
                cb.setContents(data, types);
            }
        });
        //粘贴菜单项
        final MenuItem pasteItem = new MenuItem(menu, SWT.PUSH);
        pasteItem.setText("粘贴");
        //注册粘贴菜单项事件
        pasteItem.addSelectionListener(new SelectionAdapter() {
            //当选中该菜单项时
            public void widgetSelected(SelectionEvent e) {
                //将字符串从剪贴板中取出
                String string = (String) (cb.getContents(TextTransfer.getInstance()));
                //若不为空, 则插入到文本框中
                if (string != null)
```

```

        content.insert(string);
    }
});
//为菜单注册事件
menu.addListener(new MenuAdapter() {
    //当显示菜单时
    public void menuShown(MenuEvent e) {
        //获得选中的文本
        String selection = content.getSelectionText();
        //如果没有选中任何文本，则将复制菜单项置为不可用
        copyItem.setEnabled(selection.length() > 0);
        //检查剪贴板是否支持文本数据类型
        TransferData[] available = cb.getAvailableTypes();
        boolean enabled = false;
        for (int i = 0; i < available.length; i++) {
            if (TextTransfer.getInstance().isSupportedType(available[i])) {
                enabled = true;
                break;
            }
        }
        //如果支持，设置粘贴菜单项为可用状态
        pasteItem.setEnabled(enabled);
    }
});
content.setMenu(menu);

shell.setSize(300, 150);
shell.open();
while (!shell.isDisposed()) {
    if (!display.readAndDispatch())
        display.sleep();
}
//剪贴板调用的是系统资源，使用完需手工释放掉
cb.dispose();
display.dispose();
}
}

```

通过这个剪贴板程序，可以总结出使用剪贴板时应该注意的问题，有以下几点：

- 设置剪贴板内容的方法是：setContents(Object[] data, Transfer[] dataTypes)，注意第一个 data 数组的数据类型要与 dataTypes 数据类型相符合，否则运行时会出现报错。

例如：

```

Clipboard clipboard = new Clipboard(display);
String textData = "Hello World";
String rtfData = "{\\rtf1\\b\\i Hello World}";
TextTransfer textTransfer = TextTransfer.getInstance();
RTFTransfer rtfTransfer = RTFTransfer.getInstance();
Transfer[] transfers = new Transfer[]{textTransfer, rtfTransfer};

```

```
Object[] data = new Object[]{textData, rtfData};  
clipboard.setContents(data, transfers);
```

- ❑ 获取剪贴板内容的方法与设置剪贴板内容的过程相反，使用的方法是：getContents (Transfer transfer,int clipboards)。该方法返回的对象是 Object。例如：

```
String textData = (String)clipboard.getContents(textTransfer);  
if (textData != null) System.out.println("Text is "+textData);  
String rtfData = (String)clipboard.getContents(rtfTransfer);  
if (rtfData != null) System.out.println("RTF Text is "+rtfData);
```

- ❑ 清空剪贴板的方法是：clearContents()。
- ❑ 获得剪贴板支持的数据格式的方法是：getAvailableTypes()。
- ❑ 最后值得注意的是，剪贴板也是本地系统的资源，使用完后需要释放。

10.8 本章小结

通过本章的学习，读者应该掌握如何在 SWT 中对控件实现拖放效果。拖放的关键是处理好拖放源和拖放目标之间的关系，以及在拖放过程中数据的处理。有关如何自定义传输数据类型的内容比较复杂，如果读者想要深入这方面的知识，可学习一下 Eclipse 的源代码部分中对拖放的支持部分。



第 11 章 SWT 线程

本章将讲述 SWT 中所涉及的线程问题，SWT 在底层设计时不允许其他的线程访问 UI 线程。这样设计的好处是提高了程序运行的安全性，但这却增加了对线程调用的复杂度。

11.1 线程概述

线程是 Java 基础知识中很重要也很复杂的部分，首先回顾一下 Java 中有关线程的知识。

11.1.1 什么是线程

线程 (Thread) 是计算程序运行的最小单位。在一个应用程序中可以同时运行多个线程。以一个简单的 HelloWorld 程序为例，具体说一下什么是线程。读者应该对 HelloWorld 程序非常熟悉了，代码如下：

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello");  
        System.out.println("World!");  
    }  
}
```

虽然该程序只有两行代码，首先打印出一个简单的字符串“Hello”，然后再打印出“World!”，但其实在程序运行的背后，至少存在两个线程。其中一个线程负责调用 main() 方法，这是 Java 虚拟机规定的程序调用的入口方法，当 main() 方法结束后，该线程也就结束了。另外一个线程是 Java 特有的垃圾处理线程，大家都知道，Java 的垃圾回收机制会自动地回收不需要的对象。

下面具体看一下这个 main 线程，main() 程序中的代码很简单，一瞬间就结束了。那么现在要想让这个 main 线程等待一段时间再运行，如何实现呢？就是说开始先打印“Hello”，5 秒钟过后再打印“World!”。修改后的程序代码如下：

```
public static void main(String[] args) {  
    System.out.println("Hello");  
    try {  
        Thread.sleep(5000);  
    } catch (InterruptedException e) {  
        e.printStackTrace();  
    }  
}
```



```
System.out.println("World!");  
}
```

这样程序运行后，读者就可以看到运行结果了，首先输出“Hello”，然后等待 5 秒后再输出“World! ”。这里想说明的是，对于一个线程，可以有多种状态，比如该程序中让线程进入了等待状态。当然线程还有挂起、运行等状态。线程的知识很复杂也很重要，如果读者对线程不熟悉，希望能阅读一下这方面的资料。

11.1.2 创建线程的两种方式

在以上的程序中，main 线程是 Java 虚拟机自动调用的，那么能不能编写程序来自动创建一个线程呢？答案是肯定的。Java 的设计者在设计之初就将对线程的操作考虑进来，所以在 Java 程序中创建线程是非常容易的。Java 中创建线程有两种方式：一种是实现 runnable 接口，一种是继承 Thread 对象并覆盖 run 方法。下面详细看一下这两种方式。

1. 继承 Thread

首先创建一个类 ExtendThread，该类继承自 Thread 类，并且覆盖其 run() 方法，该线程的功能是打印出一段字符串。代码如下：

```
public class ExtendThread extends Thread{  
    //覆盖 Thread 类中的 run()方法  
    public void run() {  
        System.out.println("Extend Thread");  
    }  
}
```

其中 run() 方法体中的程序是线程执行的程序。再写一个调用该程序的方法，负责启动这个线程。调用该线程的 ThreadTestDrive 类代码如下：

```
public class ThreadTestDrive {  
    public static void main(String[] args) {  
        System.out.println("Hello");  
        //创建线程对象  
        ExtendThread extendThread = new ExtendThread ();  
        //启动线程后开始调用线程对象中的 run()方法体  
        extendThread.start();  
        System.out.println("World!");  
    }  
}
```

该测试程序只是比 11.1.1 节的 HelloWorld 程序多了调用另外一个线程的代码。如果读者只在代码中看，可能认为程序运行后输出的结果顺序是：“Hello”、“Extend Thread”、“World! ”，事实上，这只是一种结果。还有一种结果是：“Hello”、“World!”、“Extend Thread”。

下面来看一下程序运行的原理，首先程序开始运行后，main 线程开始启动，main 线程运行 System.out.println("Hello") 代码后，开始创建 extendThread 线程对象，并启动这个线程。接着 main 线程继续运行 System.out.println("World!")，打印出“World”，而不是等待

extendThread 线程，打印出 Extend Thread 后再运行 System.out.println("World!")。

对 main 线程来说，执行完 System.out.println("World!")语句，这个线程就结束了，而不会影响到 extendThread 线程的运行。为了更好地说明该问题，改动一下 ExtendThread 中的 run()方法，让 extendThread 线程能运行更长的一段时间，如下：

```
public class ExtendThread extends Thread{
    public void run() {
        while(true)
            System.out.println("Extend Thread");
    }
}
```

这样改动后，ExtendThread 线程将会永远进行下去，就是说当 main 线程结束后，ExtendThread 将会永久地持续下去，程序会一直输出 Extend Thread 字符。当然，这是非常危险的，为了能在运行一段时间后让该线程自动停止，再进行一下改进，设置一个计数器，每次输出字符将计数器加 1，输出 10 次后，停止运行。改进后的代码如下：

```
public class ExtendThread extends Thread {
    int count = 0;
    public void run() {
        while (count < 10){
            System.out.println("Extend Thread");
            count++;
        }
    }
}
```

这样，再运行 ThreadTestDrive 类，main 线程运行结束后，extendThread 线程还在继续输出 Extend Thread 字符串，直到输出 10 次为止，结束线程。这更能说明，main 方法中的程序运行并不需要等待 extendThread 线程程序结束后再执行，而是这两个线程同时存在，具体执行哪个线程要靠底层的操作系统来调度了。

通过以上几个线程的程序，可以用图 11.1 来表示在运行程序中 main 线程和 extendThread 线程运行的生命周期。

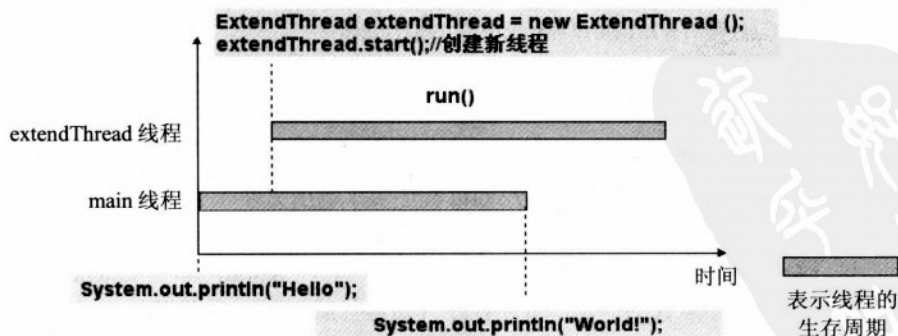


图 11.1 线程的生存周期示意图

从图 11.1 中可以清楚地看出，两个线程是同时存在的，就是说其中任何一个线程都不需要等待另一个线程结束才会继续运行。

2. 实现 Runnable 接口

第一种方法是通过继承的方式来实现的，那么这种方式创建的类就具有 Thread 的属性和方法。但是以下所说的一种情况该如何实现线程呢？有这样一个类 Child，它继承自父类 Parent，该类的代码如下：

```
public class Child extends Parent {  
}
```

因为 Java 是不能够多继承的，不能够同时继承 Parent 类和 Thread 类，要想使该类继承自 Parent 类又要具有线程的特性，此时就需要使用第二种方法：让该类实现 Runnable 接口。实现 Runnable 接口后的 Child 代码如下：

```
public class Child extends Parent implements Runnable{  
    int count = 0;  
    public void run() {  
        while (count < 10) {  
            System.out.println("Implements Thread");  
            count++;  
        }  
    }  
}
```

当然实现接口就要实现接口中的 run() 方法，这样一个具有线程特性的类就创建完成了，该线程的作用是打印出 10 次 “Implements Thread” 字符。下面来看一下如何启动这种方式创建线程。修改一下 ThreadTestDrive.java 中的代码如下：

```
public class ThreadTestDrive {  
    public static void main(String[] args) {  
        System.out.println("Hello");  
        //创建线程对象  
        ExtendThread extendThread = new ExtendThread ();  
        //启动线程后开始调用线程对象中的 run()方法体  
        extendThread.start();  
        //创建 Child 对象，Child 类实现了 Runnable 接口  
        Child child = new Child();  
        //创建线程对象，将 Child 对象作为构造参数  
        Thread implementThread = new Thread ( child );  
        implementThread.start();  
        System.out.println("World!");  
    }  
}
```

从以上的代码中可以看出，第二种方式创建线程时将实现 Runnable 接口的对象作为构造参数，这样就使该类具有了线程的属性和方法。

11.2 SWT 中的 UI 线程

SWT 作为一种桌面程序，比普通的 Java 程序要多一个 UI 线程，UI 线程负责不断地画出显示的 UI 控件，当然这个 UI 线程还要负责事件的处理。什么是事件呢？例如单击按钮或是按下键盘，系统都会生成一个事件放在事件队列中，即接下来 UI 线程按顺序处理队列中的事件。SWT 中 Display 对象就是一个 UI 线程，并且负责管理队列中的事件。

以下代码，读者并不陌生，在之前使用的 SWT 程序中都用到过，下面来仔细分析一下它的详细情况。

```
//当窗口未释放时
while (!shell.isDisposed()) {
    //如果 display 对象事件队列中没有了等待的事件，就让该线程进入等待状态
    if (!display.readAndDispatch())
        display.sleep();
}
```

可以这样理解 UI 线程：当程序启动后，如果用户不进行任何操作，该 UI 线程就进入了等待状态。一旦触发了某个事件，比如说单击了某个按钮或是按下了键盘上的按键，这时在事件队列中就等待了一个事件，此时 UI 线程就处理队列中的事件，直至队列中的事件全部处理完毕，又恢复了睡眠状态。处理事件的过程也就是响应用户操作的过程。

这就产生了一个问题，当某一个队列中的事件是一个非常耗时的事件时，比如说是检索文件或者是查询大量数据的数据库时，这时，用户就需要长时间的等待。所以在这种情况下，就需要为长时间处理的程序单独开辟出一个线程来执行，不影响 UI 线程继续处理其他事件了，这样给用户的感受就是，虽然后台运行着程序，但也不会影响界面上的操作。

11.3 其他线程访问 UI 线程

理解了线程的基本知识，再重新看一下 9.7 节中进度条的示例程序。这个程序是在开始运行窗口时就在后台启动了一个线程，这个后台线程每隔 0.1 秒更新运行一次设置滚动条的值。可以看到在 run() 方法体中有一段代码读者可能有疑问，重新加注释后的代码如下：

```
//创建一个线程，该线程每 0.1 秒更新一次滚动条的值
Runnable runnable = new Runnable() {
    public void run() {
        //线程运行的主体
        for (int i = minimus; i < maximum; i++) {
            try {
                //让线程睡眠 0.1 秒
                Thread.sleep(100);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

错误

```

    }
    //如果使用以下一行代码更新滚动条的值，运行时会出现 Invalid thread access 运行
    //progressBar.setSelection(progressBar.getSelection() + 1);
    //让 UI 线程更新滚动条的值
    display.asyncExec(new Runnable() {
        //这也是一个线程，该线程的功能是更新滚动条的值，一瞬间就结束了
        //并且这个线程是被 UI 线程调用的
        public void run() {
            if (progressBar.isDisposed())
                return;
            progressBar.setSelection(progressBar.getSelection() + 1);
        }
    });
}
};
//启动这个线程
new Thread(runnable).start();

```

在设置滚动条的 `progressBar` 的值时，为什么不能直接使用 `progressBar.setSelection(progressBar.getSelection() + 1)` 代码来直接设置滚动条的值呢？这是因为滚动条对象是 UI 界面上的控件，它是由 UI 线程创建的。若要访问 UI 界面上的对象必须通过 UI 线程来访问，就是在非 UI 线程中调用 UI 对象是不允许的，这是出于线程安全的考虑。正因为如此，只能通过另一种方式来更新进度条的值，解决方案就是需要再开辟一个线程，专门更新滚动条的值，这个线程交给 UI 线程来调用。

Display 对象中负责调用其他线程的方法有以下 3 种：

- ❑ `asyncExec(Runnable runnable)`：异步启动新的线程。所谓异步就是，UI 线程不会等待 `runnable` 对象执行结束后再继续进行，就是说 UI 线程可以和 `runnable` 对象所在的线程同时运行。
- ❑ `syncExec(Runnable runnable)`：同步启动新的线程。所谓同步就是，UI 线程会等待 `runnable` 对象执行结束后才会继续进行，当 `runnable` 对象是耗时大的线程时，尽量不要采用此种方式。另外，对于该种方式创建的线程可通过 `getSyncThread()` 方法获得线程对象。
- ❑ `timerExec(int milliseconds, Runnable runnable)`：指定一段时间再启动新的线程。用此方法创建的线程，将会在指定的时间后再启动线程。当然用此方法创建的线程启动后，与 UI 线程是异步的。如果指定的时间为负数，将不会按时启动线程。

另外 Display 对象中，与 UI 线程相关的方法如下所示：

- ❑ 获得当前的 UI 线程对象的方法：`getThread()`，返回 `Thread` 对象。
- ❑ 使 UI 线程处于休眠状态：`sleep()`。
- ❑ 唤醒 UI 线程：`wake()`。

11.4 改进的进度条

当然，虽然之前显示的进度条是使用线程进行的，但是在实际对线程的使用过程中，不可能创建一个线程之后就不再管理这个线程，一定要有一个可以控制线程的方法，即在适当的时候可以启动这个线程，在适当的时候又可以取消这个线程。

所以这里对前一个进度条的程序进行了改造，加上了一个“开始”和“取消”按钮。当单击“开始”按钮时，可以创建一个线程；当单击“取消”按钮时，又可以结束这个线程，这样就达到了对线程的控制，不会让线程自生自灭了。增加了“开始”和“取消”按钮后的进度条如图 11.2 所示。



图 11.2 改进后的进度条效果图

该程序实现的关键代码如下：

ProgressBarComplex.java

```
package com.fengmanfei.ch11;

import org.eclipse.swt.*;
import org.eclipse.swt.events.*;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.widgets.*;

public class ProgressBarComplex {
    // 表示线程的状态
    static boolean bCancel = false;
    public static void main(String[] args) {
        Display display = new Display();
        final Shell shell = new Shell(display);
        shell.setText("ProgressBar");
        shell.setLayout(new GridLayout(2, false));
        // “开始”和“取消”按钮
        final Button start = new Button(shell, SWT.NONE);
        start.setText("开始");
        Button cancel = new Button(shell, SWT.NONE);
        cancel.setText("取消");
        // 创建滚动条实例
        final ProgressBar progressBar = new ProgressBar(shell, SWT.HORIZONTAL);
        GridData data = new GridData();
        data.horizontalSpan = 2;
        data.grabExcessHorizontalSpace = true;
        progressBar.setLayoutData(data);
        progressBar.setMaximum(100); // 设置最大值
        progressBar.setMinimum(0); // 设置最小值
        final int maximum = progressBar.getMaximum(); // 获取最大值
```



```

final int minimus = progressBar.getMinimum();// 获取最小值
// 为开始按钮注册事件
start.addListener(new SelectionAdapter() {
    // 当单击"开始"按钮时
    public void widgetSelected(SelectionEvent e) {
        // 首先设置“开始”按钮为不可用状态
        start.setEnabled(false);
        // 创建更新进度条的线程
        Runnable runnable = new Runnable() {
            public void run() {
                for (int i = minimus; i < maximum; i++) {
                    try {
                        Thread.sleep(100);
                    } catch (InterruptedException e) {
                    }
                    // 注意在更新进度条时加上了判断线程状态的条件
                    shell.getDisplay().asyncExec(new Runnable() {
                        public void run() {
                            if (progressBar.isDisposed())
                                return;
                            // 如果此时取消了线程，将进度条设置为初始状态
                            if (bCancel) {
                                progressBar.setSelection(0);
                            }
                        }
                    });
                    progressBar.setSelection(progressBar.getSelection() + 1);
                }
            }
        });
        // 如果此时取消了线程，结束该循环，这个线程也就结束了
        // 并重置线程状态
        if (bCancel) {
            bCancel = false;
            break;
        }
    }
});
// 启动这个线程
new Thread(runnable).start();
}

// 注册“取消”按钮事件
cancel.addListener(new SelectionAdapter() {
    // 当单击“取消”按钮时
    public void widgetSelected(SelectionEvent e) {
        // 如果此时线程在执行，则取消线程并将开始按钮置为可用
        if (!bCancel) {
            bCancel = true;
            start.setEnabled(true);
        }
    }
});

```

```

    }
}
});
shell.pack();
shell.open();
while (!shell.isDisposed()) {
    if (!display.readAndDispatch())
        display.sleep();
}
display.dispose();
}
}

```

修改后的程序应该注意以下几个问题:

- ❑ 线程是否取消是根据 bCancel 变量来判断的, 这里只能将 bCancel 设置为 static 类型而不能将其设置为 final 类型, 因为一旦设置为 final 类型, bCancel 的值就不会改变了, 请读者一定要注意。
- ❑ 若读者想把 bCancel 变量设置为 final 类型, 可以将其类型设置为 boolean 数组型, 如:

```
final boolean[] bCancel= new boolean[1];
```

这样通过设置 bCancel [0] = true 也可以达到同样的效果, 因为虽然 final 型的基本类型数据是不可改变值的, 但对于引用型变量是可以重新赋值的, 而数组恰是引用型的。

11.5 多线程程序设计

通过以上的讲解, 读者已经对线程有了一定的认识, 有了上述对线程知识的基础, 再理解多线程的程序就不难了。在 9.9.10 节中, 学习了如何创建一个带有进度条的表格, 当然那里举的示例程序是静态的, 并没有涉及多线程的运用。下面对这个带进度条的表格进行改进, 再结合上个示例程序中对线程的控制, 改进一下该程序。如图 11.3 所示为使用多线程后, 带进度条的表格程序运行后的示意图。

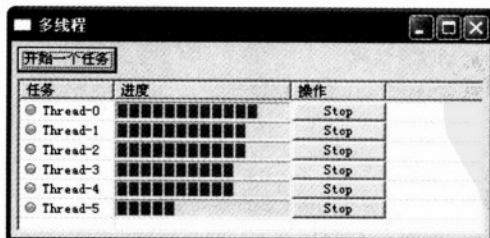


图 11.3 多线程带进度条的表格

之前使用的程序几乎都是在 main() 方法创建的, 这样根本没有发挥 Java 的面向对象思想, 而在实际的运用中肯定是用到大量的面向对象的类, 读者应该熟悉 OO 的编程方式。所以这里就采用面向对象的方式, 读者可以仔细体会一下与之前写的程序有什么不同之处。

(1) 编写一个 MutiTaskTestDrive 类，作为该系统的入口。该类比较简单，代码如下：

MutiTaskTestDrive.java

```
package com.fengmanfei.ch11;

import org.eclipse.swt.widgets.Display;
import com.fengmanfei.util.ImageFactory;

public class MutiTaskTestDrive {
    public static void main(String[] args) {
        Display display = Display.getDefault();
        MutiTaskGUI mutiTask = new MutiTaskGUI();
        mutiTask.getShell().open();
        while (!mutiTask.getShell().isDisposed()) {
            if (!display.readAndDispatch()) {
                display.sleep();
            }
        }
        ImageFactory.dispose();
        display.dispose();
    }
}
```

(2) MutiTaskGUI 类为主窗口类，也就是放置表格的窗口，该类具体的代码如下：

MutiTaskGUI.java

```
package com.fengmanfei.ch11;

import org.eclipse.swt.SWT;
import org.eclipse.swt.events.SelectionAdapter;
import org.eclipse.swt.events.SelectionEvent;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.widgets.Table;
import org.eclipse.swt.widgets.TableColumn;

public class MutiTaskGUI {
    private Shell shell = null;
    private Table table = null;
    public MutiTaskGUI() {
        //构造方法中调用初始化窗口的方法
        init();
    }
    //初始化窗口方法
    public void init() {
        shell = new Shell();
    }
}
```

```

shell.setLayout(new GridLayout());
shell.setText("多线程");
Button bt = new Button ( shell , SWT.NONE);
bt.setText("开始一个任务");
// 创建表格对象
table = new Table(shell, SWT.BORDER);
table.setLayoutData( new GridData(SWT.FILL,SWT.FILL,true,true));
table.setHeaderVisible(true);
table.setLinesVisible(true);
String[] header = new String[]{"任务","进度","操作"};
// 创建表头
for (int i = 0; i < 3; i++) {
    TableColumn col = new TableColumn(table, SWT.NONE);
    col.setText( header[i] );
}
//设置表头宽度
table.getColumn(0).setWidth(80);
table.getColumn(1).setWidth(150);
table.getColumn(2).setWidth(80);
shell.pack();
//注册创建任务按钮事件
bt.addSelectionListener( new SelectionAdapter(){
    //当单击创建一个任务按钮时
    public void widgetSelected(SelectionEvent e) {
        //首先创建一个 Task 对象
        Task task = new Task ( table );
        //然后在表格中添加一行
        task.createTableItem();
        //最后启动该任务，该任务为一个线程
        task.start();
    }
});
}
//获得和设置属性的 getter 和 setter 方法
public Shell getShell() {
    return shell;
}
public void setShell(Shell shell) {
    this.shell = shell;
}
public Table getTable() {
    return table;
}
public void setTable(Table table) {
    this.table = table;
}
}

```

该类需要注意的地方是，“开始一个任务”按钮事件的处理。当单击该按钮时，就创

建一表格中的一行，并且启动一个线程。添加表格中的一行和启动线程是使用 Task 对象来完成的。

(3) Task 类继承自 Thread，并覆盖了 run()方法，具有线程的特性。Task 类具体实现的代码如下：

Task.java

```
package com.fengmanfei.ch11;

import org.eclipse.swt.SWT;
import org.eclipse.swt.custom.TableEditor;
import org.eclipse.swt.events.SelectionAdapter;
import org.eclipse.swt.events.SelectionEvent;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.ProgressBar;
import org.eclipse.swt.widgets.Table;
import org.eclipse.swt.widgets.TableItem;

import com.fengmanfei.util.ImageFactory;

//该 Task 类继承自 Thread，并且覆盖了 run()方法
public class Task extends Thread {
    //该类的一些属性
    private Table table = null;
    //是否停止的标志
    private boolean done = false;
    //声明进度条对象
    private ProgressBar bar = null;
    private int min = 0;
    private int max = 100;
    public Task(Table table) {
        this.table = table;
    }
    //创建表格中的一行
    public void createTableItem() {
        TableItem item = new TableItem(table, SWT.NONE);
        item.setText(this.getName());
        item.setImage(ImageFactory.loadImage(table.getDisplay(),
ImageFactory.PROGRESS_TASK));
        // 创建一个进度条
        bar = new ProgressBar(table, SWT.NONE);
        bar.setMinimum(min);
        bar.setMaximum(max);
        // 创建一个可编辑的表格对象
        TableEditor editor = new TableEditor(table);
        editor.grabHorizontal = true;
        editor.grabVertical = true;
        // 将进度条绑定到第二列中
        editor.setEditor(bar, item, 1);
    }
}
```

```

//重新创建一个可编辑的表格对象
editor = new TableEditor(table);
editor.grabHorizontal = true;
editor.grabVertical = true;
//创建一个按钮
Button stop = new Button(table, SWT.NONE);
stop.setText("Stop");
editor.setEditor(stop, item, 2);
stop.addSelectionListener(new SelectionAdapter() {
    //当停止按钮按下时, 设置停止标记 true
    public void widgetSelected(SelectionEvent e) {
        if (!isDone())
            setDone(true);
    }
});
}
//线程方法体, 与前面单个的进度条的程序类似
public void run() {
    for (int i = min; i < max; i++) {
        try {
            Thread.sleep(100);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        table.getDisplay().asyncExec(new Runnable() {
            public void run() {
                if (bar.isDisposed())
                    return;
                bar.setSelection(bar.getSelection() + 1);
            }
        });
        //如果停止, 则结束该线程
        if (isDone()) {
            break;
        }
    }
}
//获得和设置属性的 getter 和 setter 方法
public Table getTable() {
    return table;
}
public void setTable(Table table) {
    this.table = table;
}
public boolean isDone() {
    return done;
}
public void setDone(boolean done) {
    this.done = done;
}

```



```
}  
}
```

Task 类中的 run() 方法体中的代码与之前单个进度条的处理方式类似。从以上的程序代码中可以看出, 类中大量使用了 bean 的方式, 也就是通过一些 getter 和 setter 方法来设置和访问类的属性, 又将常用的操作封装为方法, 这才是涉及结构合理的类。

11.6 本章小结

通过本章的学习, 读者应该对线程的知识有了一定的了解, 本章只是介绍了很简单的线程编程。对线程的程序设计来说, 需要注意的地方还很多, 比如说如何防止死锁和资源冲突等问题。如果读者在实际的项目中运用线程进行编程, 千万要倍加小心。

另外, 对 SWT 程序中特有的线程——UI 线程也应该有一个大致的了解。UI 线程由 Display 对象负责管理, 是 SWT 程序运行的核心。最值得注意的是, 如果在非 UI 线程中调用界面上的控件对象是会在运行期间报错, 所以 Display 对象使用 asyncExec(Runnable runnable) 和 syncExec(Runnable runnable) 方法更新界面的控件对象。



第 12 章 SWT 系统资源

本章将讲述 SWT 中系统资源的有关知识，是 SWT 程序中很重要的基础知识。系统资源包括颜色（Color）、字体（Font）、光标（Cursor）和图像（Image）等，对于这些本地系统资源，在使用后一定要释放掉，否则会严重影响系统的整体性能。

12.1 系统资源概述

开发桌面应用程序常常会遇到占用系统资源的问题，这与 GUI 类库的实现机制以及整体的实现思想有关，因此需要在应用程序中很小心地处理这些容易产生效率问题的地方。

12.1.1 什么是系统资源

什么是系统资源呢？系统资源一般是指颜色（Color）、字体（Font）、光标（Cursor）和图像（Image）等，SWT 中创建的各种控件也属于系统资源。当在 SWT 中创建了系统资源，那么也就创建了对应的本地资源。例如，创建一个 Display 对象后，本地也就创建了一个 Display 对象的引用，在这种情况下要显式地将其释放掉。

```
Display display = new Display();  
//使用完 Display 对象后释放掉...  
display.dispose();
```

再例如，创建一个颜色对象，那么也就在系统的底层资源里分配了一个颜色资源，这种情况下也需要释放其资源。

```
//创建红色对象  
Color color = new Color(display,255,0,0);  
//创建了颜色系统资源，也要在使用完时释放  
color.dispose();
```

📌 注意：如以下代码所创建的颜色对象，不需要释放掉。


```
//获得系统中的红色对象  
Color color = display.getSystemColor(SWT.COLOR_RED);
```

这是因为，这种方式是从底层平台获得的颜色资源，这个资源并不是我们创建的，只是被我们引用了，其他的对象有可能正在使用或者将要使用这个资源，释放这样的资源会带来不可预知的严重后果，所以千万不要释放掉像这种方式创建的资源。

细心的读者可能会有疑问了，Java 不是有垃圾回收机制吗，就不能够自动释放这些资

源吗？答案是否定的。Java 的垃圾回收只能回收 Java 虚拟机中的对象，而对本地操作系统的对象是不能够自动释放的，所以对本地资源的释放一定要显式地释放掉。

SWT 中需要释放资源的机制也正是最具争议的原因之一。既然选择了 SWT，所以一定要在使用资源时倍加小心。

 **小知识：**Java 之父 James Gosling 曾对 SWT 发表过一些看法，他不太赞同 SWT 的原因之一就是因为，SWT 中资源释放的处理机制比较复杂，有悖于 Java 自动回收垃圾的机制。

12.1.2 释放资源的原则

1. 通过 new 创建的资源一定要显式地释放

如果使用了 new 创建的资源一定要显式地释放，因为用 new 方式创建对象的同时也在本地创建了资源对象，例如以下为使用 new 创建的字体对象：

```
Font font = new Font( display, "Courier", 10, SWT.NORMAL);  
//使用后需释放掉  
font.dispose();
```

但是如果该字体对象不是通过 new 方式获得，而是获得其他字体的引用，例如以下为从某一个控件中获得的字体资源：

```
Font font = control.getFont ();
```

这种情况下就不需要显式释放。因为如果释放了，那么 control 对象中就不能再使用该字体资源了。

总之，对使用 new 方式创建的资源一定要记得释放。

2. 释放了父控件也就释放了子控件

各种控件 Shell、Button、Text 等对象也是系统资源，使用完后也需要释放掉。但当一个窗口上的控件很多时，不可能一一将控件释放掉，可以想象这是多么庞大的工作量。所以 SWT 提供了一个机制，就是释放了父控件也就是释放了子控件。

例如以下代码，有一个窗口（Shell）对象，窗口中有一个按钮（Button）。释放窗口后也就隐式地释放了按钮对象，而不再需要再显式地释放按钮对象。

```
Shell shell = new Shell( display , SWT.NONE);  
Button button = new Button( shell, SWT.NONE);  
//使用完释放 shell，也就释放了 button，不需要再调用 button.dispose()  
shell.dispose();
```

虽然释放遵循该原则，但仍需要注意：该规则只是适合 Widget 的子类。但是如果应用程序中使用了字体、颜色、图像、鼠标等系统资源，仍需要在程序中显式地调用 dispose() 方法释放。

例如在以上的窗口中设置了图标，创建了图像对象，虽然释放了窗口，但仍需释放图像资源。

```
Shell shell = new Shell( display , SWT.NONE);
Image icon = new Image( display , "F:\\icon.gif");
Button button = new Button( shell, SWT.NONE);
//使用完释放 shell, 也就释放了 button, 不需要再调用 button.dispose()
shell.dispose();
//仍需要释放图像资源
icon.dispose();
```

3. 释放了控件也就释放了控件所设置的菜单

就是说如果为一个控件通过 `control.setMenu()` 方法设置了该控件的菜单, 当释放了 `control` 对象时, 不仅释放了该对象的子控件, 同时也隐式地释放了该控件的 `Menu` 对象。

```
Shell shell = new Shell( display , SWT.NONE);
Button button = new Button( shell, SWT.NONE);
Menu menu = new Menu( shell , SWT.POP_UP);
//...创建菜单项
button.setMenu( menu);
shell.dispose();
//释放了 button 对象就释放了 menu 对象, 不需要调用 menu.dispose()
```

总之, 释放了父类控件也就释放了子类控件和该控件所设置的菜单资源, 但控件中使用的诸如图像、字体等资源仍需显式地释放掉。

12.1.3 访问资源的原则

SWT 中不允许访问已经释放了的控件。一旦一个控件在底层的操作系统中被释放掉了, 那么在使用该控件的 Java 对象时就会抛出 “Widget is disposed” 异常, 所以在使用控件的 Java 对象时要判断该资源是否已经释放掉了。

判断控件是否被释放的方法是使用控件对象的 `isDisposed()` 方法, 如果控件已经被释放则返回 `true`, 否则返回 `false`。

所以就不难理解在 11.4 节中, 为什么在更新进度条之前, 先要判断一下进度条控件是否被释放的原因了。判断进度条是否被释放的代码如下:

```
//如果进度条被释放了, 则不更新滚动条
if (progressBar.isDisposed())
return;
```

12.1.4 何时释放资源

既然知道了创建的资源是需要释放的, 那在什么情况下释放资源是最合适的时机呢? 当然这里讨论的情况是针对需要显式释放的资源。

1. 最简单的情况: 使用完就释放

例如, 现在要为按钮设置字体。当按钮使用完字体资源后就立即释放掉。

```
//创建字体对象
```

```
Font font = new Font (display, "Courier", 10, SWT.NORMAL);
Button button = new Button( shell, SWT.NONE);
button.setText("ddd");
button.setFont( font );
//该字体对象只被按钮使用，按钮使用完即可释放掉
font.dispose();
```

但现在又将为窗口也设置此字体对象，则需要在窗口使用完字体对象后释放。

```
//创建字体对象
Font font = new Font (display, "Courier", 10, SWT.NORMAL);
Button button = new Button( shell, SWT.NONE);
button.setText("ddd");
button.setFont( font );
//窗口对象也使用了该字体
shell.setFont(font);
//窗口和按钮都使用完才可释放掉
font.dispose();
```

2. 在释放控件事件 `widgetDisposed()` 的方法中释放资源

在调用控件的 `dispose()` 方法时可以触发 `widgetDisposed()` 事件，所以可以在该事件中释放资源。例如上个例子中，在 Shell 窗口的 `widgetDisposed()` 事件中释放资源代码如下：

```
//创建字体对象
final Font font = new Font (display, "Courier", 10, SWT.NORMAL);
Button button = new Button( shell, SWT.NONE);
button.setText("ddd");
//按钮和窗口分别使用了字体对象
button.setFont(font);
shell.setFont(font);
//注册窗口释放时所触发的事件
shell.addDisposeListener( new DisposeListener(){
    //当窗口释放时
    public void widgetDisposed(DisposeEvent e) {
        //如果此时字体未释放，则释放字体
        if (!font.isDisposed())
            font.dispose();
    }
});
```

总之，不管使用哪种机制释放资源，一定要在使用完后的某个时机释放掉，否则会严重影响系统的整体性能。

12.2 颜色 (Color)

SWT 中涉及颜色的对象都在 `org.eclipse.swt.graphics` 包，其中有颜色对象 (Color) 和 RGB 对象等。其中 Color 类继承图如图 12.1 所示。

12.2.1 系统颜色

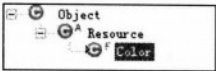


图 12.1 Color 类继承图

系统颜色是 SWT 内置的一些常见的颜色，可以使用 org.eclipse.swt.SWT 类中封装的常量获得。使用 display 对象的 `getSystemColor(int id)` 方法获得。例如，创建一个蓝色的系统颜色的代码如下：

```
Color color = display.getSystemColor( SWT.COLOR_BLUE);
```

注意：此种方法使用的颜色不需要释放掉。

表 12.1 显示了 SWT 中可使用的颜色常量及其对应的 RGB 的值。

表 12.1 SWT 中可使用的颜色常量

样 式 常 量	Red 红色值	Green 绿色值	Blue 蓝色值	说 明
SWT.COLOR_BLACK	0	0	0	黑色
SWT.COLOR_DARK_RED	128	0	0	深红色
SWT.COLOR_DARK_GREEN	0	128	0	深绿色
SWT.COLOR_DARK_YELLOW	128	128	0	深黄色
SWT.COLOR_DARK_BLUE	0	0	128	深蓝色
SWT.COLOR_DARK_MAGENTA	128	0	128	深紫红色
SWT.COLOR_DARK_CYAN	0	128	128	深蓝绿色
SWT.COLOR_DARK_GRAY	128	128	128	深灰色
SWT.COLOR_GRAY	192	192	192	灰色
SWT.COLOR_RED	255	0	0	红色
SWT.COLOR_GREEN	0	255	0	绿色
SWT.COLOR_YELLOW	255	255	0	黄色
SWT.COLOR_BLUE	0	0	255	蓝色
SWT.COLOR_MAGENTA	255	0	255	紫红色
SWT.COLOR_CYAN	0	255	255	蓝绿色
SWT.COLOR_WHITE	255	255	255	白色

除了这些常用的颜色外，还提供了系统的标题颜色等，这些颜色会根据操作系统用户的设置而显示不同的颜色。表 12.2 显示了内置的系统颜色的常量。

表 12.2 系统当前使用的颜色常量表

样 式 常 量	说 明
SWT.COLOR_INFO_FOREGROUND	提示 (ToolTip) 的前景色
SWT.COLOR_INFO_BACKGROUND	提示 (ToolTip) 的背景色
SWT.COLOR_TITLE_FOREGROUND	标题栏的前景色
SWT.COLOR_TITLE_BACKGROUND	标题栏的背景色
SWT.COLOR_TITLE_BACKGROUND_GRADIENT	标题栏的背景渐近色
SWT.COLOR_TITLE_INACTIVE_FOREGROUND	处于非激活状态时标题栏的前景色

续表

样 式 常 量	说 明
SWT.COLOR_TITLE_INACTIVE_BACKGROUND	处于非激活状态时标题栏的背景色
SWT.COLOR_TITLE_INACTIVE_BACKGROUND_GRADIENT	处于非激活状态时标题栏的背景渐近色
SWT.COLOR_WIDGET_DARK_SHADOW	控件阴影区域的颜色
SWT.COLOR_WIDGET_NORMAL_SHADOW	控件正常情况下阴影区域的颜色
SWT.COLOR_WIDGET_LIGHT_SHADOW	控件浅色阴影的颜色
SWT.COLOR_WIDGET_HIGHLIGHT_SHADOW	高亮显示阴影区域的颜色
SWT.COLOR_WIDGET_FOREGROUND	默认控件的前景色
SWT.COLOR_WIDGET_BACKGROUND	默认控件的背景色
SWT.COLOR_WIDGET_BORDER	默认控件的边框景色
SWT.COLOR_LIST_FOREGROUND	列表的前景色
SWT.COLOR_LIST_BACKGROUND	列表的背景色
SWT.COLOR_LIST_SELECTION	列表选中时的背景色
SWT.COLOR_LIST_SELECTION_TEXT	列表选中时的文字颜色

ⓘ注意：使用这些颜色时会根据当前用户设置的系统颜色不同而不同。

12.2.2 RGB 颜色

颜色一般是由三原色也就是红（Red）、绿（Green）和蓝（Blue）3 种颜色通过不同的组合形成的。计算机使用的颜色中，每个三原色的取值范围是 0~255 之间的任意一个数值。例如，创建一个黑色的 RGB 颜色的代码为：

```
RGB black= new RGB(0,0,0)
```

白色的 RGB 颜色代码为：

```
RGB black= new RGB(255,255,255)
```

当然这只是创建了 RGB 对象，还需要创建颜色对象后才能使用。创建颜色对象的代码如下：

```
RGB black= new RGB(0,0,0);
Color color = new Color( display , black );
```

或者

```
Color color = new Color( display , 0,0,0 );
```

这两种方式创建的颜色的方式是一样的。

ⓘ注意：这种方式创建的颜色在使用完后需要释放掉。

以下为 Color 类的常用方法：

- ❑ 获得蓝色的值，返回 int: getBlue()。
- ❑ 获得绿色的值，返回 int: getGreen()。

- ❑ 获得红色的值，返回 int: `getRed()`。
- ❑ 获得颜色的 RGB 对象: `getRGB()`。

12.3 字体 (Font)

字体与颜色类似也属于系统资源，SWT 中涉及字体的类有 `Font` 和 `FontData`。它们的关系与 `Color` 和 `RGB` 类的关系类似，`FontData` 对象携带了一些字体的信息，包括字体名、字体大小等。但要使用字体对象还要使用 `Font` 类将 `FontData` 对象包装一下。`Font` 类的继承关系如图 12.2 所示。

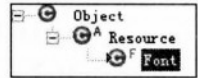


图 12.2 `Font` 类继承图

`Font` 类的构造方法有以下 3 种：

- ❑ `Font(Device device, String name, int height, int style)`。例如：

```
Font font = new Font(display, "Courier", 10, SWT.NORMAL);
```

其中，`Courier` 表示字体的名称，10 表示字体的大小，`SWT.NORMAL` 表示正常字体，另外还可以指定加粗 `SWT.BOLD` 或倾斜 `SWT.ITALIC`。

- ❑ `Font(Device device, FontData fd)`。例如：

```
FontData data = new FontData("Courier", 10, SWT.NORMAL);
Font font = new Font(display, data);
```

这种方式创建的字体与第一种方法创建的字体对象相同。

- ❑ `Font(Device device, FontData[] fds)`。例如：

```
FontData[] data = new FontData[2];
data[0] = new FontData("Courier", 10, SWT.NORMAL);
data[1] = new FontData("宋体", 10, SWT.NORMAL);
Font font = new Font(display, data);
```

此种方法可以同时创建多个字体对象。

创建字体对象后可以使用 `getFontData()` 方法，将创建的字体信息取出来，该方法返回为 `FontData[]` 数组类型。

下面来看一下 `FontData` 类，该类也很简单，主要携带了字体的名称、大小和样式等信息。它的构造方法有以下 3 种：

- ❑ `FontData()`。
- ❑ `FontData(String string)`。
- ❑ `FontData(String name, int height, int style)`。

另外，该类除了 `getter` 和 `setter` 方法外，还有一个设置本地语言的方法 `setLocale(String locale)`。最后，除了显示的创建字体外，也可以获得系统默认的字体，方法是使用 `Display` 对象的 `getSystemFont()`。例如：

```
Font font = display.getSystemFont();
```

⚠注意：这种方式获得的系统字体是不需要释放的。

12.4 光标 (Cursor)

光标也是常用的系统资源之一，SWT 中使用 `org.eclipse.swt.graphics.Cursor` 类来表示光标。`Cursor` 类的继承关系如图 12.3 所示。

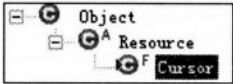


图 12.3 Cursor 类继承关系图

创建光标对象有两种，一种是使用系统的光标，一种是自定义的光标。下面分别来看一下如何使用这两种光标。

1. 系统光标

创建系统光标时要使用 `Cursor(Device device, int style)` 构造方法。例如，创建一个手型的光标代码如下：

```
Cursor cursor = new Cursor( display , SWT.CURSOR_HAND);
```

其中光标可选的样式常量和光标的效果如图 12.4 所示。

2. 自定义光标

自定义光标可以使用以下两种构造方法来创建：

- ❑ `Cursor(Device device, ImageData source, ImageData mask, int hotspotX, int hotspotY)`。
- ❑ `Cursor(Device device, ImageData source, int hotspotX, int hotspotY)`。

其中，`source` 参数表示光标使用的光标文件，`hotspotX` 与 `hotspotY` 参数表示光标进入到有效区域内的 X 和 Y 坐标值，`mask` 代表光标滤镜效果。下面就举一个使用指定光标图片作为光标的小程序，该程序运行后的效果如图 12.5 所示。

CURSOR_APPSTARTING		CURSOR_IBEAM		CURSOR_SIZE	
CURSOR_ARROW		CURSOR_NO		CURSOR_SIZENESW	
CURSOR_CROSS		CURSOR_SIZEALL		CURSOR_SIZENS	
CURSOR_HAND		CURSOR_SIZEE		CURSOR_SIZENW	
CURSOR_HELP		CURSOR_SIZEN		CURSOR_SIZESW	
CURSOR_SIZES		CURSOR_SIZESE		CURSOR_SIZEWE	
CURSOR_SIZEWE		CURSOR_UPARROW		CURSOR_WAIT	

图 12.4 光标样式常量与其对应的效果图



图 12.5 自定义光标

该程序实现的主要代码如下：

ImageCursor.java

```
Display display = new Display();
ImageData sourceData = new ImageData ("F:\\icon\\cursor.gif");
Cursor cursor = new Cursor(display , sourceData ,10, 10);
```

```

Shell shell = new Shell(display);
shell.setSize(150, 150);
shell.open();
shell.setCursor(cursor);
while (!shell.isDisposed()) {
    if (!display.readAndDispatch())
        display.sleep();
}
cursor.dispose();
display.dispose();

```

12.5 图像 (Image)

图像是系统中最常用的资源，一般有两种用途，一种是用作控件显示的图标，通常使用 `setImage(Image image)` 方法；另外一种是由于浏览图片，通常这种情况下一般使用画布类 (Canvas) 来显示图片。第一种使用图片的方法很简单，但第二种图片的方法就比较复杂了，可能会涉及图像透明度、拉伸、反转等效果。Image 类的继承示意图如图 12.6 所示。

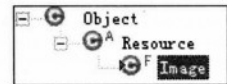


图 12.6 Image 类继承关系图

12.5.1 画布类 (Canvas)

在深入了解图像之前，先学习一下画布类，该类继承自 Composite 类，可以看作是一种面板类。顾名思义，画布是能够在上面画的面板类。可以在画布上画任何线条、图形，当然也包括图像，在这里主要是使用画布来显示图像。图 12.7 显示了该类的继承关系图。

Canvas 类与之前学习的面板类非常类似，在这里想利用一个小程序来说明如何使用画布类来显示图片。该程序运行后的效果如图 12.8 所示。

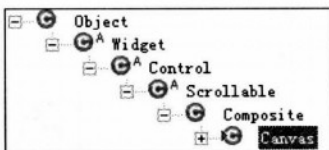


图 12.7 Canvas 类继承关系图

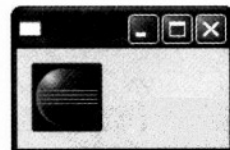


图 12.8 画布显示图片程序效果图

该程序的代码如下：

CanvasSample.java

```

package com.fengmanfei.ch12;

import org.eclipse.swt.events.*;
import org.eclipse.swt.graphics.*;
import org.eclipse.swt.layout.FillLayout;

```



```
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.SWT;

public class CanvasSample {
    private Image image = null;
    private Shell shell = null;
    private Canvas canvas = null;
    public CanvasSample() {
        createContent();
    }
    public void createContent() {
        shell = new Shell();
        shell.setLayout(new FillLayout());
        // 创建一个图像对象
        image = new Image(shell.getDisplay(), getClass().getResourceAsStream ("eclipse48.gif"));
        // 创建一个画布对象
        canvas = new Canvas(shell, SWT.NONE);
        // 注册画布的画图事件
        canvas.addPaintListener(new PaintListener() {
            public void paintControl(PaintEvent e) {
                // 在画布上显示图像，图像在画布中的坐标是距左边 10 个像素，距上边 10 个像素
                e.gc.drawImage(image, 10, 10);
            }
        });
        shell.setSize(200, 150);
    }
    public Canvas getCanvas() {
        return canvas;
    }
    public void setCanvas(Canvas canvas) {
        this.canvas = canvas;
    }
    public Image getImage() {
        return image;
    }
    public void setImage(Image image) {
        this.image = image;
    }
    public Shell getShell() {
        return shell;
    }
    public void setShell(Shell shell) {
        this.shell = shell;
    }
    public static void main(String[] args) {
        Display display = Display.getDefault();
        CanvasSample cSample = new CanvasSample();
        cSample.getShell().open();
    }
}
```

```

        while (!cSample.getShell().isDisposed()) {
            if (!display.readAndDispatch())
                display.sleep();
        }
        cSample.getImage().dispose();
        display.dispose();
    }
}

```

通过上述程序，总结出使用画布类时应该注意以下方面：

- ❑ 创建画布时，该程序中使用的样式是 `SWT.NONE`，另外还可以选择使用的样式有 `SWT.NO_REDRAW_RESIZE`、`SWT.NO_BACKGROUND` 和 `SWT.NO_MERGE_PAINTS` 等。
- ❑ 若要在画布上绘制图像，需在画布的画图事件 `PaintEvent` 中完成。其中画图事件中可以使用 `event.gc` 获得一个 `GC` 对象。`GC` (`Graphics Context`) 对象是一个绘图对象，可以绘制线条、矩形、椭圆形和图像等。在这里我们是利用它的绘制图像的功能，通过使用 `drawImage(Image image, int x, int y)` 方法来实现的。`GC` 对象的其他绘图功能将在下文有关绘图一章中详细讲述。

12.5.2 图像类 (Image)

图像类的来源有两种，一种是通过程序来绘制的，在报表统计图中，这种图像最为常见，一般这种图像是与数据密切相关的；另一种是使用图像编辑工具生成的图片，这种图片一般被保存为扩展名为 `.BMP`、`.ICO`、`.JPEG`、`.GIF` 和 `.PNG` 格式。本章所涉及的图片资源都是已经保存了的图片，有关如何创建绘制的图片将在绘图一章详细讲述。在前面的程序中，也涉及一些图片创建方法，在这里就总结一下创建图片的各种方法：

- ❑ `Image(Device device, String filename)`: `filename` 表示图片的地址。例如：

```
Image image= new Image (display , "F:\\icon\\cursor.gif");
```

这种方法是最常见的创建图像的方法。

- ❑ `Image(Device device, InputStream stream)`: `stream` 表示图片文件的数据流。例如：

```
Image image = new Image(display, getClass().getResourceAsStream("eclipse48.gif"));
```

这里需要注意的是，`getClass().getResourceAsStream("eclipse48.gif")` 方法获得的数据流文件，是与该 `.java` 类同文件夹的 `eclipse48.gif` 文件的数据流。当然通过数据流创建图片的好处是，当图片是以数据流的方式保存在数据库中时，就可以直接从数据库中读出数据流来创建图像对象了。

- ❑ `Image(Device device, Image srcImage, int flag)`: 这种方法可以简单地将图片转换成不同风格的图像对象。例如，将一个图片转换成灰色的图片代码如下：

```
Image image = new Image(display, getClass().getResourceAsStream("eclipse48.gif"));
Image grayImage = new Image( display , image , SWT.IMAGE_GRAY);
```

创建一个灰色的图像后的效果如图 12.9 所示。

另外可选的样式还有 SWT.IMAGE_DISABLE, 如图 12.10 所示。若使用 SWT.IMAGE_COPY 样式, 则创建一个与该图片相同的图片对象。

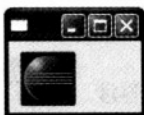


图 12.9 SWT.IMAGE_GRAY 样式



图 12.10 SWT.IMAGE_DISABLE 样式

□ Image(Device device, ImageData data): 使用 ImageData 对象来创建图片对象。例如:

```
ImageData data = new ImageData("F:\\icon\\cursor.gif");
Image image = new Image(display, data);
```

有了前面学习的 Font 与 FontData 的关系就不难理解 Image 和 ImageData 的关系了, ImageData 对象保存的是有关图片的信息, 包括图片所包含的颜色、大小、类型等, 要使用图片还要利用 Image 将 ImageData 对象包装一下。ImageData 类将在 12.5.3 节讲述。

综上所述, 有以上几种常见的创建图像对象的方法, 当创建完图像对象后, 可以使用 getImageData() 方法获得图像对象的详细信息, 该方法返回 ImageData 对象。下面就具体来看一下 ImageData 这个类的详细情况。

12.5.3 图像数据类 (ImageData)

ImageData 对象可以在转化成 Image 对象之前对图片进行包装加工, 比如说可以放大图片, 设置图片的透明度等, 相当于一个简单的处理图片工具。创建 ImageData 对象常用的方法有以下两种:

- ImageData(InputStream stream): 以图片文件的数据流作为参数。
- ImageData(String filename): 以图片的路径作为参数。

这两种方法与创建 Image 对象的方法类似, 但是没有 Display 对象, 因为 ImageData 是与设备无关的对象, 只保存了图片的信息。下面来看一下该类的属性和常用的方法:

- height、width 表示图片的高和宽, type 表示图片的类型。支持的类型有 SWT.IMAGE_BMP、SWT.IMAGE_BMP_RLE、SWT.IMAGE_GIF、SWT.IMAGE_ICO、SWT.IMAGE_JPEG 和 SWT.IMAGE_PNG。
- data: 图像的二进制数组, 类型为 byte[]。
- getRGBs(): 获得该图像所包含的 RGB 颜色, 返回类型为 RGB[] 数组。
- scaledTo(int width, int height): 放大图片的方法。注意, 放大后返回一个新的 ImageData 对象, 而不改变当前的图片大小。例如, 使用以下代码可以创建一个放大原来图片两倍的图片。

```
ImageData data = image.getImageData();
Image i1 = new Image(e.display, data);
e.gc.drawImage(i1, 10, 10);
```

```

ImageData doubleImage = data.scaledTo( data.width*2,data.height*2 );
Image i2 = new Image(e.display, doubleImage);
e.gc.drawImage(i2, 80, 10);
i1.dispose();
i2.dispose();

```

放大图片后的效果如图 12.11 所示。

- ❑ 另外还可以设置图片的透明度、通道值等，这些内容涉及对颜色和处理图片的理论，这里就不多作介绍了。有兴趣的读者可以阅读一下这方面的资料。

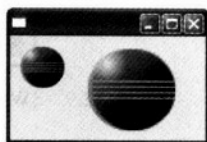


图 12.11 放大后的图片

12.5.4 保存图像类 (ImageLoader)

对于通过 ImageData 加工后的图像可以使用 ImageLoader 对象保存为新文件。例如，将上述放大的图片保存为 “doubleImage.gif”，代码如下：

```

ImageData doubleImage = data.scaledTo( data.width*2,data.height*2 );
ImageLoader loader = new ImageLoader ();
//设置图片的数据
loader.data = new ImageData[]{doubleImage};
loader.save("F:\\icon\\doubleImage.gif",SWT.IMAGE_GIF);

```

经过以上转换，即将放大为原来两倍的图片保存到指定的文件中了。

⚠注意：保存图片时要注意只能保存有效的图片格式，否则会抛出 “ERROR_UNSUPPORTED_FORMAT” 异常。

另外该类除了可以保存图片外，也可以加载图片。常用加载图片的方法有：

- ❑ load(InputStream stream): 指定输入流加载。
- ❑ load(String filename): 指定文件路径加载。

这两种方法都可以装载一个图像对象，并且返回一个 ImageData[] 数组对象。值得注意的是，该类使用的都为 ImageData[] 数组，原因是如果图片格式为 gif 格式时，可能为一个动态的图片，学过一些设计的读者应该知道，一个动态的图片其实就是由多个图片构成，每个图片可以理解为一帧，所以使用数组就是为了保存动态图片的。

对于像 gif 的图片还可以设置 repeatCount 属性，表示多帧图片所重复的次数。默认为 1，若设置为 0，则表示永远循环下去。

12.5.5 Eclipse 的图标

通常 Eclipse 平台直接使用的图标文件都存放在以下的包中，通常可以直接拿来使用。

- ❑ org.eclipse.debug.ui/icons.
- ❑ org.eclipse.pde.ui/icons.
- ❑ org.eclipse.jdt.ui/icons.

- ☐ org.eclipse.vcm.ui/icons。
- ☐ org.eclipse.team.ui/icons。
- ☐ org.eclipse.ant.ui/icons。
- ☐ org.eclipse.help.ui/icons。
- ☐ org.eclipse.ui/icons。
- ☐ org.eclipse.ui.views/icons。
- ☐ org.eclipse.ui.console/icons。

为了使读者能获得全部的 Eclipse 的图标文件,笔者在此特将 Eclipse 的图标文件打包附在本书配套的光盘中,读者可以直接获取这些图标。

12.6 SWT 绘图

SWT 中的绘图功能是通过 org.eclipse.swt.graphics 包中的 GC 对象来实现的,任何实现了 Drawable 的类都可以进行绘制。如图 12.12 所示为实现了 Drawable 接口的类。

从图中不难发现,各种控件类和图像类都是可以绘制的。下面就来看一下如何使用 GC 对象进行绘图,

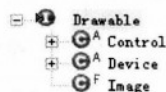


图 12.12 实现了 Drawable 接口类继承关系示意图

12.6.1 使用绘制对象的方法

通常使用 GC 对象时有两种方法,一种是使用 GC 的构造方法来创建 GC 对象,另一种方法是通过 addPaintListener 对象来绘制。

1. 使用构造方法

GC 的构造方法有两个:

- ☐ GC(Drawable drawable): 其中 drawable 为绘图接口对象,如图 12.12 所示的子类都可以作为构造的对象传入。
- ☐ GC(Drawable drawable, int style): 其中 style 可以为 SWT.LEFT_TO_RIGHT 或是 SWT.RIGHT_TO_LEFT 或是 SWT.NONE。

例如,以下所示的代码为将 Shell 窗口作为一个绘图对象构造一个 GC 对象:

```
GC gc = new GC(shell);
gc.dispose();
```

2. 在绘图监听器对象中绘制

对于支持绘图功能的控件可以通过 addPaintListener 方法注册绘图监听器,来绘制图形。例如,以下代码所示为 Shell 窗口对象注册了绘图监听器。

```
shell.addPaintListener(new PaintListener() {
    public void paintControl(PaintEvent e) {
```



```
e.gc.drawOval(0,0,100,100);
}
});
```

12.6.2 绘制线条

通过 GC 对象可以绘制各种线条，只需要使用它的方法即可。这里绘制线条的方法如下所示：

- `drawLine(int x1, int y1, int x2, int y2)`: 绘制一条直线，其中 `x1`、`y1` 为直线的一个点坐标，`x2`、`y2` 为直线另一个点的坐标。例如，以下代码在画布上绘制一条直线：

```
e.gc.drawLine(20,20,100,20);
```

画出来的效果如图 12.13 所示。

- `drawPolyline (int[] pointArray)`: 绘制多边形，其中 `pointArray` 为各个顶点的 `x` 和 `y` 坐标。例如，以下代码为绘制一个多边形，运行后效果如图 12.14 所示。

```
e.gc.drawPolyline(new int[] { 50,10,75,45,25,45 });
```

- `drawPolygon(int[] pointArray)`: 绘制多边形，其中 `pointArray` 为各个顶点的 `x` 和 `y` 坐标。例如，以下代码为绘制一个多边形，运行后效果如图 12.15 所示。

```
e.gc.drawPolyline(new int[] { 50,10,75,45,25,45 });
```

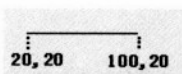


图 12.13 画直线

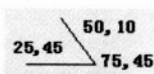


图 12.14 画多边形

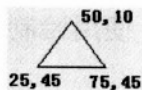


图 12.15 画多边形

- `drawRectangle(int x, int y, int width, int height)`: 绘制矩形，其中 `x`、`y` 为矩形的左上角顶点坐标，`width` 为矩形的宽，`height` 为矩形的高。例如，以下代码为绘制一个矩形，运行后的效果如图 12.16 所示。

```
e.gc.drawRectangle(5,20,100,50);
```

- `drawOval(int x, int y, int width, int height)`: 绘制椭圆，其中 `x`、`y` 为椭圆左上角的坐标，`width` 表示椭圆的宽，`height` 表示椭圆的高，如果宽和高相等，则绘制出圆形。例如，以下代码为绘制的一个椭圆，代码运行后效果如图 12.17 所示。

```
e.gc.drawOval( 5,20, 100,50);
```

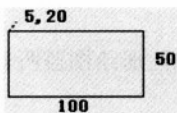


图 12.16 绘制矩形

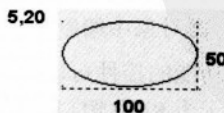


图 12.17 绘制椭圆

- ❑ `drawRoundRectangle(int x, int y, int width, int height, int arcWidth, int arcHeight)`: 绘制圆角矩形, 其中 `x`、`y`、`width` 和 `height` 的意义与绘制矩形的意义相同, `arcWidth` 与 `arcHeight` 为圆角弧度, 如图 12.18 所示为这两个值所代表的意义。

例如以下代码为绘制了圆角矩形的代码, 代码运行后效果如图 12.19 所示。

```
e.gc.drawRoundRectangle( 5,20, 100,50,25,15);
```

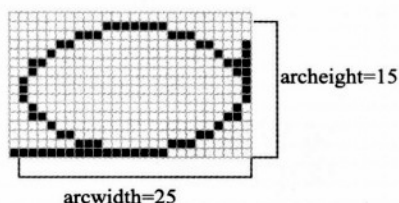


图 12.18 圆角参数的意义

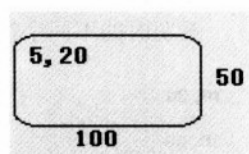


图 12.19 圆角矩形

- ❑ `drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)`: 绘制一段弧, 其中 `startAngle` 为弧起始点的角度, 可以是 $0^\circ \sim 360^\circ$ 任意范围的值, 0° 表示坐标平面中 `x` 的正半轴, `arcAngle` 所转过的弧度, 若为正值, 则表示逆时针, 负值表示顺时针。例如, 以下代码绘制了一个 120° 的弧。运行后的效果如图 12.20 所示。

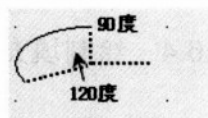


图 12.20 绘制弧

- ❑ `drawPoint(int x, int y)`: 绘制一个点, 其中 `x` 表示该点的横坐标, `y` 表示该点的纵坐标。
- ❑ `setLineStyle(int lineStyle)`: 设置线条的样式, 常量及其效果如表 12.3 所示。

表 12.3 设置线条样式

样 式 常 量	效 果 图
SWT.LINE_SOLID	—————
SWT.LINE_DOT
SWT.LINE_DASH	-----
SWT.LINE_DASHDOT	- . - . - .
SWT.LINE_DASHDOTDOT	- . - . - . - .

- ❑ `setLineWidth(int lineWidth)`: 设置线条的粗细, 以像素为单位。

12.6.3 绘制字符

通过 GC 对象不仅可以绘制线条, 也可以将文本绘制出来。使用的方法有以下几种:

- ❑ `drawString(String string, int x, int y)`: 其中 `string` 为要显示的字符串, `x` 表示字符串所在的横坐标、`y` 表示字符串所在的纵坐标。例如, 以下代码为绘制两个字符串, 代码运行后效果如图 12.21 所示。

```
Font font = new Font(e.display,"Arial",14,SWT.BOLD | SWT.ITALIC);
e.gc.drawString("This is a String",30,20);
e.gc.setForeground(e.display.getSystemColor(SWT.COLOR_GREEN));
e.gc.setFont(font);
e.gc.setBackground(e.display.getSystemColor( SWT.COLOR_DARK_GRAY ));
e.gc.drawString("This Is A Sting",30,50);
font.dispose();
```

- ❑ `drawString(String string, int x, int y, boolean isTransparent)`: 绘制字符串, 其中 `isTransparent` 表示是否制定该文本透明显示。例如, 图 12.22 显示了设置透明与不透明时的不同字符串效果。

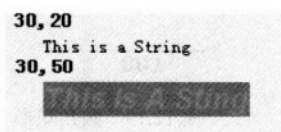


图 12.21 绘制字符串

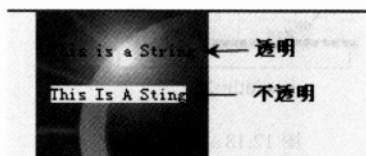


图 12.22 字符串的透明与不透明效果

12.6.4 绘制填充图形

除了绘制矩形、椭圆形这些线条外, 也可以绘制一些填充图形。例如, 图 12.23 所示为填充的矩形。所涉及绘制填充图形的方法有以下几种, 其中使用的一些参数与绘制线条时的方法类似。

- ❑ `fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)`: 填充扇形。
- ❑ `fillOval(int x, int y, int width, int height)`: 填充椭圆形。
- ❑ `fillPolygon(int[] pointArray)`: 填充多边形。
- ❑ `fillRectangle(int x, int y, int width, int height)`: 填充矩形。
- ❑ `fillRoundRectangle(int x, int y, int width, int height, int arcWidth, int arcHeight)`: 填充圆角矩形。
- ❑ `fillGradientRectangle(int x, int y, int width, int height, boolean vertical)`: 填充渐近的矩形。其中 `vertical` 表示渐近的方式是否垂直。例如, 以下代码为填充了渐近色后的矩形, 代码运行后的效果如图 12.24 所示。`vertical` 改为 `false` 则效果如图 12.25 所示。

```
e.gc.setBackground(e.display.getSystemColor(SWT.COLOR_BLUE));
e.gc.setForeground(e.display.getSystemColor(SWT.COLOR_CYAN));
e.gc.fillGradientRectangle(5,20,100,50,false);
```

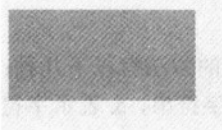


图 12.23 填充的矩形



图 12.24 垂直渐近色的矩形



图 12.25 水平渐近色的矩形

12.6.5 绘制图像

GC 对象另一个重要的方法是可以绘制图像，将图像绘制到指定的位置。有关绘制图像的方法如下所示：

- ❑ `drawImage(Image image, int x, int y)`: 其中 `image` 为所绘制的图像，`x` 为图像左上角的横坐标，`y` 为纵坐标。
- ❑ `drawImage(Image image, int srcX, int srcY, int srcWidth, int srcHeight, int destX, int destY, int destWidth, int destHeight)`: 其中，`srcX`、`srcY`、`srcWidth` 和 `srcHeight` 表示原始图像中的横坐标、纵坐标、宽度和高度。`destX`、`destY`、`destWidth` 和 `destHeight` 表示显示在控件中的横坐标、纵坐标、宽度和高度。

例如，以下代码为显示的图像运行后的效果，如图 12.26 所示。

```
gc.drawImage(image,20,0,95,82,5,5,95,82);
```

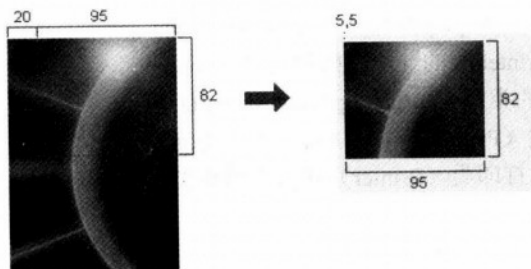


图 12.26 绘制图像

12.7 本章小结

通过本章的学习，读者应该了解什么是系统资源了，而且千万要记得在使用完系统资源后释放掉。同时，释放资源是要遵循一定的原则。另外还学习了常见的系统资源，如颜色、字体、光标和图像等。图像是比较复杂的系统资源，其中还涉及一些图片处理的理论问题，这部分平时用的也不是很多，但需要读者稍微了解一下。

正因为对资源的管理在很大程度上影响着整体的性能，所以要有一个很好的资源管理机制。幸运的是，JFace 和 Eclipse 中对系统资源都有一套很好的管理机制。在以后的章节中还要详细地讲解，这里就先不多说了。

另外，SWT 对绘图的支持也很丰富，本章只介绍了最简单的 GC 对象，还只停留在最简单的应用，如果读者想进一步了解 SWT 绘图的知识，可以阅读一些 Draw 2D 和 Eclipse 提供的图形开发框架 GEF 的知识。

第 13 章 SWT 的高级应用

通过本章的学习，读者将掌握 SWT 的一些高级应用，包括对打印的支持、使用系统的应用程序打开文件、如何使用 Windows 的 OLE 程序。而且在了解了 SWT 桌面应用外，SWT 程序还可以应用到掌上设备和 Web 开发中。

13.1 打印支持

打印是常用的输出功能，SWT 通过 `Printer` 类可以实现打印的功能。`Printer` 类与 `Display` 对象都属于设备的一种，都继承自 `Device` 类。`Printer` 类的继承图如图 13.1 所示。

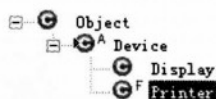


图 13.1 `Printer` 类继承关系图

SWT 中涉及打印的类都在 `org.eclipse.swt.printing` 包中，包括打印对话框类（`PrintDialog`）（对话框在讲述对话框时已经讲述过）、打印类（`Printer`）和打印数据类（`PrinterData`）。下面就来详细看一下这两个类。

13.1.1 打印类（`Printer`）和打印数据类（`PrinterData`）

有了之前 `Image` 和 `ImageData` 学习的基础，很容易理解 `Printer` 和 `PrinterData` 的关系，`PrinterData` 只是保存一些打印设置的参数，与设备无关。而 `Printer` 是一种打印设备，打印时的一些设置要通过 `PrinterData` 来获得。

`PrinterData` 对象通常通过打印对话框来获得，具体用法请参阅 9.8.6 节。当然也可以通过 `Printer.getDefaultPrinterData()` 获得默认的 `PrinterData` 对象。这里着重介绍一下打印类 `Printer`。创建打印类有两种方法：

- ❑ `Printer()`：创建默认的打印类。例如：`Printer printer = new Printer();`。
- ❑ `Printer(PrinterData data)`：通过 `PrinterData` 对象创建打印类。例如：

```
PrinterData data = Printer.getDefaultPrinterData();
Printer printer = new Printer( data );
```

那么有了打印对象，打印的过程如何呢？一般打印一个文件，首先要启动一个打印任务，使用 `startJob(String jobName)` 方法可以启动打印任务，并且可以设置打印的名称。然后调用 `startPage()` 开始打印一页，当该页打印完成后则调用 `endPage()` 完成该页。若此时还需要打印多页，则重复调用 `startPage()`、`endPage()`，直到所有的打印页都结束，最后调用 `endJob()` 完成此次打印任务。若在打印的过程中要取消打印，则调用 `cancelJob()` 方法即可。

为了使读者更清楚地了解打印的整个过程，图 13.2 显示了整个打印的流程图。

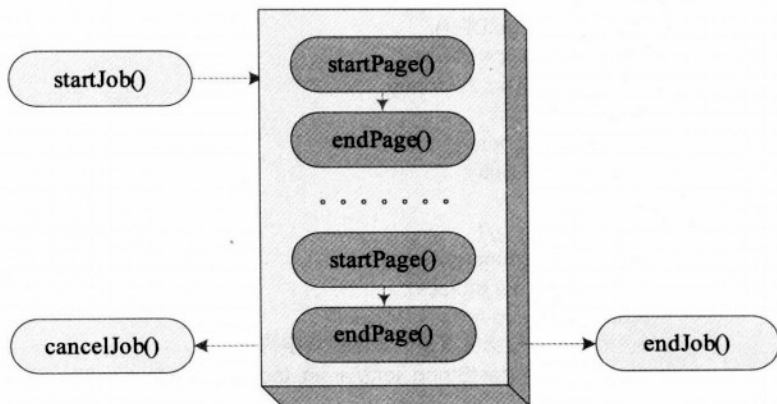


图 13.2 打印的流程图

那么了解了打印的流程，如何确定打印的内容呢？下面以一个小程序来讲述如何设定打印的内容。该小程序的功能是：打印两页文件，每页打印出不同的“Hello World!”字符。程序运行后效果如图 13.3 和图 13.4 所示。

Hello World!

图 13.3 第一页的字符串效果图

Hello World!

图 13.4 第二页的字符串效果图

实现该程序的代码如下：

SimplePrint.java

```
package com.fengmanfei.ch13;


import org.eclipse.swt.SWT;
import org.eclipse.swt.graphics.*;
import org.eclipse.swt.printing.Printer;
import org.eclipse.swt.printing.PrinterData;

public class SimplePrint {
    public static void main(String[] args) {
        PrinterData data = Printer.getDefaultPrinterData();
        if (data == null) {
            return;
        }
        Printer printer = new Printer(data);
        //开始打印任务
        if (printer.startJob("Simple Print")) {
            Color black = printer.getSystemColor(SWT.COLOR_BLACK);
            Color gray = printer.getSystemColor(SWT.COLOR_GRAY);
            Color red = printer.getSystemColor(SWT.COLOR_RED);
```

```

//计算左边距和上边距的位置
Rectangle trim = printer.computeTrim(0, 0, 0, 0);
Point dpi = printer.getDPI();
int leftMargin = dpi.x + trim.x;
int topMargin = dpi.y / 2 + trim.y;
//创建图形上下文对象
GC gc = new GC(printer);
Font font = gc.getFont();
//打印第一页
if (printer.startPage()) {
    gc.setBackground(gray);
    gc.setForeground(black);
    String testString = "Hello World!";
    //使用 GC 对象画字符串的方法显示字符
    gc.drawString(testString, leftMargin, topMargin + font.getFontData()[0].
getHeight());
    printer.endPage();
}
//打印第二页
if (printer.startPage()) {
    gc.setBackground(gray);
    gc.setForeground(black);
    String testString = "Hello World!";
    Point extent = gc.stringExtent(testString);
    gc.drawString(testString, leftMargin, topMargin + font.getFontData()[0].
getHeight());
    gc.setForeground(red);
    gc.drawRect(leftMargin, topMargin, extent.x, extent.y);
    printer.endPage();
}
//释放 GC 对象
gc.dispose();
//结束打印任务
printer.endJob();
}
printer.dispose();
}
}

```

 提示: 为了便于测试打印的效果, 建议读者安装一个 Adobe Acrobat 软件, 并设定默认的打印设备为打印成.pdf格式的文件。这样可以通过打印的 pdf 文件来查看打印效果。

通过以上的程序, 不难发现打印的内容是通过 GC 对象的 drawString()方法来输出的, 在之前讲述图像时, 提到过 GC 对象, 该对象也可以绘制字符串到指定的位置。简单地说, 打印的内容是通过绘图对象 GC 将内容绘制出来, 然后打印出来的。GC 对象是打印内容的载体。在这里只使用 GC 对象绘制字符串的方法。

另外, 在使用 Printer 对象时, 还要注意计算出页边距等值。精确地绘制出字符串的位置对打印字符很重要。下面就来看下一个打印字符更为复杂的程序例子。

13.1.2 打印程序示例概述

下面的这个程序综合了前面学习的菜单、对话框、颜色的知识，还加入了使用打印的功能。该程序的功能大致是，有一个文本框和菜单。可以通过菜单项打开一个系统文件，比如说.java 文件，文件打开后将显示在文本框中。同时也可以设置显示在文本框中文字的字体、前景色和背景色，当设置好后，可以选择打印菜单项，按照当前的格式打印出来。该程序运行后的界面效果如图 13.5 所示。



图 13.5 打印文本程序效果图

13.1.3 打印程序示例：主窗口程序

这里为了更好地让读者了解整个程序的思路，首先显示主程序的框架代码，然后再分别对调用的方法进行说明。程序实现的主程序代码如下：

PrintGUI.java

```
package com.fengmanfei.ch13;

import java.io.*;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.*;
import org.eclipse.swt.graphics.*;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.printing.*;
import org.eclipse.swt.widgets.*;

public class PrintGUI {

    private Shell sShell ;//打印的窗口对象
    private Menu menu ;//菜单对象
    private Text content ;//文本框对象
    Font font;//字体对象
    Color foregroundColor, backgroundColor;//前景色和背景色
```

```
//创建菜单的方法
private void createMenu() {
    menu = new Menu ( sShell , SWT.BAR);
    sShell.setMenuBar( menu );
    //文件菜单项
    MenuItem item = new MenuItem(menu, SWT.CASCADE);
    item.setText("文件(&F)");
    Menu fileMenu = new Menu(sShell, SWT.DROP_DOWN);
    item.setMenu(fileMenu);
    //打开菜单项
    item = new MenuItem(fileMenu, SWT.PUSH);
    item.setText("打开(&O)");
    item.setAccelerator(SWT.CTRL + 'O');
    item.addSelectionListener(new SelectionAdapter() {
        public void widgetSelected(SelectionEvent event) {
            menuOpen();//打开文件方法
        }
    });
    //选择字体菜单项
    item = new MenuItem(fileMenu, SWT.PUSH);
    item.setText("选择字体");
    item.addSelectionListener(new SelectionAdapter() {
        public void widgetSelected(SelectionEvent event) {
            menuFont();//选择字体方法
        }
    });
    //选择前景色菜单项
    item = new MenuItem(fileMenu, SWT.PUSH);
    item.setText("选择前景色");
    item.addSelectionListener(new SelectionAdapter() {
        public void widgetSelected(SelectionEvent event) {
            menuForegroundColor();//选择前景色方法
        }
    });
    //选择背景色菜单项
    item = new MenuItem(fileMenu, SWT.PUSH);
    item.setText("选择背景色");
    item.addSelectionListener(new SelectionAdapter() {
        public void widgetSelected(SelectionEvent event) {
            menuBackgroundColor();//选择背景色方法
        }
    });
    //打印菜单项
    item = new MenuItem(fileMenu, SWT.PUSH);
    item.setText("打印(@P)");
    item.setAccelerator(SWT.CTRL + 'P');
    item.addSelectionListener(new SelectionAdapter() {
        public void widgetSelected(SelectionEvent event) {
            menuPrint();//打印方法
        }
    });
}
```



```

    }
    });
    //退出菜单项
    new MenuItem(fileMenu, SWT.SEPARATOR);
    item = new MenuItem(fileMenu, SWT.PUSH);
    item.setText("&退出(E)");
    item.addSelectionListener(new SelectionAdapter() {
        public void widgetSelected(SelectionEvent event) {
            font.dispose();//释放字体资源
            foregroundColor.dispose();//释放前景色资源
            backgroundColor.dispose();//释放背景色资源
            System.exit(0);
        }
    });
}

protected void menuPrint() {...} //打印文件方法
protected void menuBackgroundColor() {...} //设置背景色方法
protected void menuForegroundColor() {...} //设置前景色方法
protected void menuFont() {...} //设置字体方法
protected void menuOpen() {...} //打开文件方法
//创建主窗口
private void createSShell() {
    sShell = new Shell();
    sShell.setText("Shell");
    sShell.setLayout(new FillLayout());
    sShell.setSize(new Point(300, 200));
    content = new Text(sShell, SWT.BORDER | SWT.MULTI | SWT.V_SCROLL | SWT.
H_SCROLL);
    createMenu();//创建菜单
}
public static void main(String[] args) {
    Display display = Display.getDefault();
    PrintGUI thisClass = new PrintGUI();
    thisClass.createSShell();
    thisClass.sShell.open();

    while (!thisClass.sShell.isDisposed()) {
        if (!display.readAndDispatch())
            display.sleep();
    }
    display.dispose();
}
}

```

这段程序读者应该不陌生，这里都是之前学习创建各种控件的方法，关键是每个菜单项的事件处理是之前所没有接触到的。

13.1.4 打印程序示例：打开文件程序

首先看打开文件的 `menuOpen()` 方法，该方法获取一个读入数据文件流，然后将数据转换为字符串显示到文本框中，这涉及 Java 类库中 IO 的操作。该方法的具体代码如下：

```
protected void menuOpen() {
    final String textString;
    //打开文件选择对话框
    FileDialog dialog = new FileDialog(sShell, SWT.OPEN);
    dialog.setFilterExtensions(new String[] {"*.java", "*.txt"});
    //获得选择的文件路径
    String name = dialog.open();
    if ((name == null) || (name.length() == 0)) return;
    try {
        //将文件读入到文本框中
        File file = new File(name);
        FileInputStream stream = new FileInputStream(file.getPath());
        try {
            Reader in = new BufferedReader(new InputStreamReader(stream));
            char[] readBuffer = new char[2048];
            StringBuffer buffer = new StringBuffer((int) file.length());
            int n;
            while ((n = in.read(readBuffer)) > 0) {
                buffer.append(readBuffer, 0, n);
            }
            textString = buffer.toString();
            stream.close();
        } catch (IOException e) {
            MessageBox box = new MessageBox(sShell, SWT.ICON_ERROR);
            box.setMessage("读文件出错:\n" + name);
            box.open();
            return;
        }
    } catch (FileNotFoundException e) {
        MessageBox box = new MessageBox(sShell, SWT.ICON_ERROR);
        box.setMessage("文件未找到:\n" + name);
        box.open();
        return;
    }
    content.setText(textString);
}
```

13.1.5 打印程序示例：设置字体和颜色程序

打开文件后可以设置文本框的字体、前景色和背景色，分别在 `menuFont()`、`menuForegroundColor()` 和 `menuBackgroundColor()` 方法中处理，这 3 个方法的原理类似，都是从对

话框中获取设置的数据，然后设置给文本框。这 3 个方法的具体代码如下：

```
protected void menuBackgroundColor() {  
    //打开颜色选择对话框  
    ColorDialog colorDialog = new ColorDialog(sShell);  
    colorDialog.setRGB(content.getBackground().getRGB());  
    RGB rgb = colorDialog.open();  
    if (rgb != null) {  
        if (backgroundColor != null) backgroundColor.dispose();  
        backgroundColor = new Color(sShell.getDisplay(), rgb);  
        //选择颜色后设置文本框的背景颜色  
        content.setBackground(backgroundColor);  
    }  
}  
//设置前景色  
protected void menuForegroundColor() {  
    //打开颜色对话框  
    ColorDialog colorDialog = new ColorDialog(sShell);  
    colorDialog.setRGB(content.getForeground().getRGB());  
    RGB rgb = colorDialog.open();  
    if (rgb != null) {  
        if (foregroundColor != null) foregroundColor.dispose();  
        foregroundColor = new Color(sShell.getDisplay(), rgb);  
        //选择颜色后设置文本框的前景色  
        content.setForeground(foregroundColor);  
    }  
}  
//设置字体  
protected void menuFont() {  
    //打开字体对话框  
    FontDialog fontDialog = new FontDialog(sShell);  
    fontDialog.setFontList(content.getFont().getFontData());  
    FontData fontData = fontDialog.open();  
    if (fontData != null) {  
        if (font != null) font.dispose();  
        font = new Font(sShell.getDisplay(), fontData);  
        content.setFont(font); //设置文本框的字体  
    }  
}
```

13.1.6 打印程序示例：打印文本的程序

最后，关键是如何处理打印过程。打印的处理是在 `menuPrint()` 方法中处理的，该方法首先调出打印对话框，然后进行打印。打印是一个很耗时的工作，所以有必要使用一个线程来后台进行处理。另外，为了能够使读者重用打印字符的类，所以在此笔者将其设计为单独的一个打印字符串文本的类，这样读者就可以轻易在别的系统中使用打印功能了。下

面来具体看一下 menuPrint()方法，如下：

```
protected void menuPrint() {
    //打开打印对话框
    PrintDialog dialog = new PrintDialog(sShell, SWT.NONE);
    PrinterData data = dialog.open();
    if (data == null) return;
    if (data.printToFile) {
        data.fileName = "print.out";
    }
    Printer printer = new Printer(data);
    //创建 TextPrinter 对象，该类是一个打印字符串类
    TextPrinter textPrinter = new TextPrinter( "Print Job", content , printer);
    textPrinter.start();
}
```

从以上程序中可以看到 TextPrinter 对象是执行打印的一个类，该类继承自 Thread 类，通过 start()方法来启动，该类结合了线程和打印的功能。该类的具体实现代码如下：

TextPrinter.java

```
package com.fengmanfei.ch13;

import org.eclipse.swt.graphics.*;
import org.eclipse.swt.printing.Printer;
import org.eclipse.swt.widgets.Text;

public class TextPrinter extends Thread {

    private Text content;//显示字符的文本框对象
    private Printer printer;//打印对象
    private GC gc;//用于绘制的图形上下文对象
    private FontData[] printerFontData;//打印的字体数据
    private RGB printerForeground, printerBackground;//打印的前景色和背景色
    private int lineHeight = 0;//打印的行高
    private int tabWidth = 0;//Tab 键的大小
    private int leftMargin, rightMargin, topMargin, bottomMargin;//上下左右的页边距
    private int x, y;//打印的行和列的值
    private int index, end;//字符当前打印的索引值和总长度
    private String textToPrint;//需要打印的字符
    private String tabs;//Tab 键
    private StringBuffer wordBuffer;//打印字符的临时变量

    public TextPrinter(String name, Text text, Printer printer) {
        super(name);
        this.content = text;
        this.printer = printer;
        init();
    }

    //初始化属性的方法
    public void init() {
```

```
textToPrint = content.getText();
printerFontData = content.getFont().getFontData();
printerForegroundColor = content.getForeground().getRGB();
printerBackgroundColor = content.getBackground().getRGB();
//默认一个 Tab 键表示 4 个空格
int tabSize = 4;
StringBuffer tabBuffer = new StringBuffer(tabSize);
for (int i = 0; i < tabSize; i++)
    tabBuffer.append(' ');
tabs = tabBuffer.toString();

//计算上下左右边距值
Rectangle clientArea = printer.getClientArea();
Rectangle trim = printer.computeTrim(0, 0, 0, 0);
Point dpi = printer.getDPI();
leftMargin = dpi.x + trim.x;
rightMargin = clientArea.width - dpi.x + trim.x + trim.width;
topMargin = dpi.y + trim.y;
bottomMargin = clientArea.height - dpi.y + trim.y + trim.height;
}

public void run() {
    //启动打印，如果不能正常启动则返回
    if (!printer.startJob("Text Print"))
        return;
    //开始打印
    print();
}

public void print() {
    //创建图形上下文对象
    gc = new GC(printer);
    //创建字体，前景色，背景色对象
    Font printerFont = new Font(printer, printerFontData);
    Color printerForegroundColor = new Color(printer, printerForegroundColor);
    Color printerBackgroundColor = new Color(printer, printerBackgroundColor);
    //将字体，前景色，背景色设置为 GC 对象
    gc.setFont(printerFont);
    gc.setForeground(printerForegroundColor);
    gc.setBackground(printerBackgroundColor);
    tabWidth = gc.stringExtent(tabs).x;
    lineHeight = gc.getFontMetrics().getHeight();
    //打印字符
    printText();
    //结束打印工作
    printer.endJob();
    //释放资源
    printerFont.dispose();
    printerForegroundColor.dispose();
    printerBackgroundColor.dispose();
    gc.dispose();
}
```



```

    }

    void printText() {
        //开始打印一页
        printer.startPage();
        wordBuffer = new StringBuffer();
        x = leftMargin;
        y = topMargin;
        index = 0;
        end = textToPrint.length();
        //循环每个字符串的每个字符
        while (index < end) {
            char c = textToPrint.charAt(index);
            System.out.println(c);
            index++;
            if (c == 0)
                continue;
            //如果字符是\n 换行符或\r 回车符
            if (c == '\n' || c == '\r'){
                //如果字符\n\r 同时出现, 则只打印一行
                if (c == '\r' && index < end && textToPrint.charAt(index) == '\n') {
                    index++;
                }
                printWordBuffer();
                newline();
            } else {
                //如果字符是 Tab 键, 则打印出字符
                if (c != '\t') {
                    wordBuffer.append(c);
                }
                if (Character.isWhitespace(c)) {
                    printWordBuffer();
                    if (c == '\t') { //Tab 键
                        x += tabWidth;
                    }
                }
            }
        }
        if (y + lineHeight <= bottomMargin) {
            printer.endPage();
        }
    }

    //打印字符, 要判断是否已经到了行的末尾, 如果到了末尾需要换行
    void printWordBuffer() {
        if (wordBuffer.length() > 0) {
            String word = wordBuffer.toString();
            int wordWidth = gc.stringExtent(word).x;
            if (x + wordWidth > rightMargin) {
                newline();
            }
            gc.drawString(word, x, y, false);
            x += wordWidth;
        }
    }
}

```



```
wordBuffer = new StringBuffer();
    }
}
//换行, 如果到了页尾, 需要打印下一页
void newline() {
    x = leftMargin;
    y += lineHeight;
    if (y + lineHeight > bottomMargin) {
        printer.endPage();
        if (index + 1 < end) {
            y = topMargin;
            printer.startPage();
        }
    }
}
}
```

理解该程序执行的过程大致是这样的: 循环每个要打印的字符, 如果字符是一个特殊的字符, 比如“`\r`”回车键、“`\n`”换行键和“`\t`”Tab 键值时要做特殊的处理, 另外打印每个字符时还要考虑是否该换行和换页等。处理字符的逻辑比较复杂, 希望读者认真阅读代码注释部分。

另外, 为了使该打印类更加灵活, 读者可以添加一个 `setter` 和 `getter` 方法, 在这里就不详细介绍了。

13.1.7 打印程序示例: 打印文件后的效果预览

该程序运行后, 打印出来的格式与文本框中的格式相同。如图 13.6 所示为打印为 PDF 格式文件后的效果图。

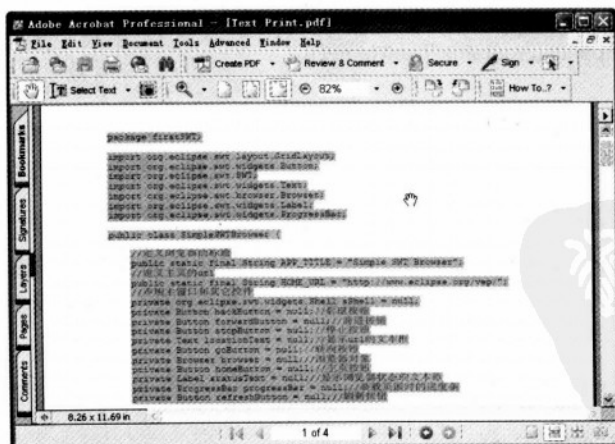


图 13.6 打印为 PDF 格式后的效果图

13.2 使用应用程序

应用程序一般是已经安装了的程序，比如说 Word、Excel 都可以称之为应用程序。一般在打开一个文件时，可以选择所打开的应用程序。例如一个.txt 文件，可以使用记事本打开，也可以使用写字板打开。SWT 中提供了访问其他应用程序的类 `Program`，该类位于 `org.eclipse.swt.program` 包中。通过该类可以打开系统中的其他应用程序。首先看一下该类中的静态方法：

- ❑ `getExtensions()`：该方法可以获得操作系统中所有支持的扩展名类型。例如，以下小程序可以输出操作系统中所有的文件扩展名。

```
public class ProgramSample {
    public static void main(String[] args) {
        Display display = new Display ();
        String[] s = Program.getExtensions();
        for (int i =0;i<s.length;i++){
            System.out.println(s[i]);
        }
        display.dispose ();
    }
}
```

⚠注意：使用 `Program` 对象时也要创建一个 `Display` 对象。

- ❑ `getPrograms()`：可以获得操作系统所安装的所有的应用程序。例如，以下代码将输出所有应用程序的名称。

```
Program[] programs =Program.getPrograms();
for (int i =0;i<programs.length;i++){
    System.out.println(programs[i].getName());
}
```

- ❑ `launch(String fileName)`：通过指定的应用程序的可执行文件的地址，可打开一个应用程序。例如，运行以下代码将打开一个浏览器，笔者的操作系统安装在 D 盘。

```
Program.launch("D:\\Program Files\\Internet Explorer\\IEXPLORE.EXE");
```

- ❑ `findProgram(String extension)`：通过文件的扩展名来获得打开该文件的应用程序。例如以下程序，先试图获得打开.pdf 格式文件的程序，如果找到了打开的程序，则使用该程序打开文件。

```
Program p = Program.findProgram (".pdf");
if (p != null)
    p.execute ("F:\\Text Print.pdf");
```

其中，`execute` 方法为打开指定目录的文件路径。

13.3 对 AWT/Swing 程序的支持

虽然说 AWT 与 SWT 程序两种桌面应用的运行机制不同,但为了使 AWT 的程序可以轻松转换到 SWT 程序中运行,SWT 提供了对 AWT 程序的支持。对 AWT 程序支持的类在 `org.eclipse.swt.awt` 包中,该包中只有一个类 `SWT_AWT` 负责对 SWT 的控件和 AWT 控件的转化。该类有两个方法,负责进行转化,如下所示:

- `static java.awt.Frame new_Frame(Composite parent)`: 该方法负责将 SWT 的面板对象转化成 AWT 的 `Frame` 对象,但使用时要注意,传入的 `parent` 对象的样式要为 `SWT.EMBEDDED`。例如,以下代码可以在 SWT 的面板中显示 AWT 的对象。

SWT_AWT_Sample.java

```
//定义一个面板类,样式设置为 SWT.EMBEDDED
Composite composite = new Composite(shell, SWT.EMBEDDED);
//将该面板对象转化成 AWT 中的 Frame 对象
java.awt.Frame awtFrame = SWT_AWT.new_Frame(composite);
awtFrame.setSize(200,150);
awtFrame.setVisible( true );
//添加一个 AWT 标签
java.awt.Label label = new Label();
label.setText("这是一个 AWT 标签");
awtFrame.add( label ,java.awt.BorderLayout.NORTH);
////添加一个 AWT 文本框
TextField textField = new TextField();
textField.setText("这是一个 AWT 文本框");
awtFrame.add( textField ,java.awt.BorderLayout.CENTER);
```

该代码运行后效果如图 13.7 所示。

- `Shell new_Shell(Display display, java.awt. Canvas parent)`: 该方法可以将一个 AWT 的 `Canvas` 对象转化成 SWT 的 `Shell` 对象。该方法使用的过程与刚才讲述的方法相反,在这里就不想多作介绍了。

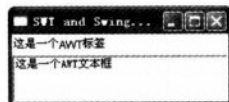


图 13.7 运行 AWT 的 SWT 程序效果图

13.4 OLE 和 ActiveX 控件的支持

什么是 OLE? 学习过 C++ 的读者可能知道,OLE (Object Link Embeded) 是指在程序之间链接和嵌入对象数据。简单地说,就是能够在 Word 中插入 Excel 表进行编辑。通过 OLE 技术可以在一个应用程序中执行其他的应用程序。

而 ActiveX 控件是 OLE 的延伸,一般用于网络。其实之前学习过的浏览器 (Browser)

对象就是利用 OLE 程序使用 IE 浏览器的功能达到的。如图 13.8 所示，即为一个在 Shell 窗口中嵌入 Word 程序的效果图，其中“文件”菜单使用的是 SWT 程序的 MenuItem 创建的。

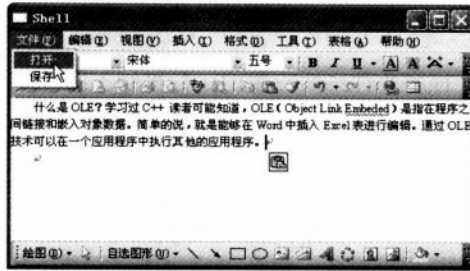


图 13.8 嵌入 Word 的 SWT 窗口

注意：有关 OLE 和 ActiveX 控件的具体描述请读者查阅微软的 CSDN 网站。

SWT 中涉及 OLE 和 ActiveX 的类都在 org.eclipse.swt.ole.win32 包中，使用最常见的是 OleFrame 类、OleClientSite 类和 OleControlSite 类。下面就来具体看一下各类的用法。

13.4.1 OLE 控件的面板类 (OleFrame)

首先看一下 OleFrame 类，该类继承自 Composite 类相当于一个放置 OLE 控件的面板。该类的继承关系图如图 13.9 所示。

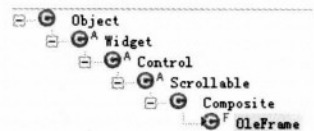


图 13.9 OleFrame 的类继承关系图

该类的主要功能有以下几点：

- 对 OLE 控件进行布局的设置，相当于一个普通的面板类。
- 可以为控件添加菜单，图 13.8 所示的程序中，文件菜单就是通过该对象创建的。有关涉及对控件菜单的方法有：
 - ◆ 设置和获取文件菜单：setFileMenus(MenuItem[] fileMenus)和 getFileMenus()。
 - ◆ 设置和获取容器菜单：setContainerMenus(MenuItem[] containerMenus)和 getContainerMenus()。
 - ◆ 设置和获取窗口菜单：setWindowMenus(MenuItem[] windowMenus)和 getWindowMenus()。
- 既然可以获取 OLE 控件的菜单，就可以对菜单项进行控制，例如设置可见和设置加速键等。

创建一个 OleFrame 对象的方法很简单，代码如下：

```
OleFrame frame = new OleFrame(sShell, SWT.NONE);
//可以为控件设置菜单项，例如：
frame.setFileMenus(fileItem);// fileItem 是已经初始化后的 MenuItem[]数组
```

13.4.2 OLE 控件类 (OleClientSite 和 OleControlSite)

有了放置 OLE 的框架就可以放置 OLE 的控件对象了。OleClientSite 对象和 OleControlSite 对象都是 OLE 控件,其中 OleClientSite 为 OLE 控件,OleControlSite 为 ActiveX 控件,因为 OleControlSite 类继承自 OleClientSite 类。这两个类的继承关系如图 13.10 所示。

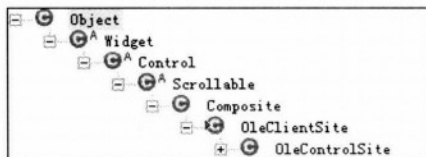


图 13.10 OleControlSite 类和 OleClientSite 类继承关系图

若想创建 OLE 对象,有两种常用的构造方法如下:

- ❑ OleClientSite(Composite parent, int style, String progId): 其中 progId 为标示应用系统的字符,例如,Word 的 progId 为“Word.Document”,Excel 的为“Excel.Sheet”,IE 的为“Shell.Explorer”。若要查看其他应用程序的 progId,可以查看系统注册表。例如,以下代码表示创建了一个 Word 的 OLE 控件:

```
OleClientSite clientSite = new OleClientSite(frame, SWT.NONE, "Word.Document");
```

- ❑ OleClientSite(Composite parent, int style, File file): file 为保存的某一个文件。用这种方法创建的 OLE 控件,系统会根据文件自动查找对应打开的应用程序。例如,以下代码可以创建一个打开.doc 文件的控件。

```
File file = new File("F:\\temp.doc");
OleClientSite clientSite = new OleClientSite(frame, SWT.NONE, file);
```

创建了一个 OLE 控件,接下来需要打开控件,才可以显示控件。使用 doVerb(int verb) 方法。其中 verb 值有以下可以选择的参数:

- ❑ OLE.OLEIVERB_PRIMARY: 打开编辑状态的 OLE 控件。
- ❑ OLE.OLEIVERB_SHOW: 显示 OLE 控件。
- ❑ OLE.OLEIVERB_OPEN: 在另外一个窗口中打开 OLE 控件。
- ❑ OLE.OLEIVERB_HIDE: 隐藏 OLE 控件。
- ❑ OLE.OLEIVERB_INPLACEACTIVATE: 不带有工具栏和菜单栏的方式。
- ❑ OLE.OLEIVERB_UIACTIVATE: 激活 OLE 的菜单栏和工具栏等。
- ❑ OLE.OLEIVERB_DISCARDUNDOSTATE: 关闭 OLE 控件。

⚠注意: OLE 的常量使用 org.eclipse.swt.ole.win32.OLE 的常量,不是 org.eclipse.swt.SWT 的常量。

例如,以下代码可以打开编辑状态的 OLE 控件:

```
clientSite.doVerb(OLE.OLEIVERB_PRIMARY);
```


这样，就可以打开 OLE 程序进行编辑了。当然 OleClientSite 对象还有一些其他很复杂的方法，比如可以监听 OLE 控件所触发的事件等，这些知识都涉及 C++ 的知识，在这里就不过多讲述了。

这里只介绍一个比较简单的功能，通常打开一个文件，经过修改后要保存。当 OLE 对打开的文件修改后，通过 isDirty() 方法可以判断是否已经修改过。如果修改后，可以使用 save(File file, boolean includeOleInfo) 方法进行保存。例如：

```
if (clientSite.isDirty()) {  
    clientSite.save(file, true)  
}
```

以上讲述的是如何创建 OLE 控件，下面来看一下如何创建 ActiveX 控件。创建 ActiveX 控件对象要使用 OleControlSite 类，该类只有一个构造方法：

OleControlSite(Composite parent, int style, String progId)：只能根据 progId 来创建，例如创建一个 Word 的 ActiveX 控件对象的代码如下：

```
OleControlSite controlSite = new OleControlSite(frame, SWT.NONE, "Word.Document");
```

其他的使用与创建 OLE 控件对象的使用方法类似。

13.4.3 OLE 程序示例

下面以一个程序来说明如何使用 OLE 控件对象。该程序的功能是：可以选择打开一个 Word 文档，然后进行编辑后保存该文档。程序运行后的效果如图 13.8 所示。以下为该程序的具体代码：

OleSample.java

```
package com.fengmanfei.ch13;  
  
import java.io.File;  
import org.eclipse.swt.SWT;  
import org.eclipse.swt.events.SelectionAdapter;  
import org.eclipse.swt.events.SelectionEvent;  
import org.eclipse.swt.graphics.Point;  
import org.eclipse.swt.layout.FillLayout;  
import org.eclipse.swt.ole.win32.OLE;  
import org.eclipse.swt.ole.win32.OleClientSite;  
import org.eclipse.swt.ole.win32.OleFrame;  
import org.eclipse.swt.widgets.*;  
  
public class OleSample {  
    private Shell sShell;  
    private MenuItem[] fileItem;//OLE 的菜单项  
    private OleClientSite clientSite;//OLE 控件对象  
    private OleFrame frame;//OLE 的面板的对象  
    private File openFile;//打开的文件
```

```

public static void main(String[] args) {
    Display display = Display.getDefault();
    OleSample thisClass = new OleSample();
    thisClass.createSShell();
    thisClass.sShell.open();
    while (!thisClass.sShell.isDisposed()) {
        if (!display.readAndDispatch())
            display.sleep();
    }
    thisClass.clientSite.dispose();
    display.dispose();
}

private void createSShell() {
    sShell = new Shell();
    sShell.setText("OLE Sample");
    sShell.setLayout(new FillLayout());
    createMenu();
    sShell.setSize(new Point(300, 200));
}

//创建 OLE 控件对象
private void createOle() {
    frame = new OleFrame(sShell, SWT.NONE);
    frame.setFileMenus(fileItem);
    if (openFile != null)
        clientSite = new OleClientSite(frame, SWT.NONE, openFile);
    clientSite.doVerb(OLE.OLEIVERB_PRIMARY);
}

private void createMenu() {
    Menu main = new Menu(sShell, SWT.BAR);

    MenuItem file = new MenuItem(main, SWT.CASCADE);
    file.setText("文件(&F)");

    Menu fileMenu = new Menu(file);
    fileItem = new MenuItem[2];
    fileItem[0] = new MenuItem(fileMenu, SWT.PUSH);
    fileItem[0].setText("打开");
    fileItem[1] = new MenuItem(fileMenu, SWT.PUSH);
    fileItem[1].setText("保存");
    file.setMenu(fileMenu);

    sShell.setMenuBar(main);

    fileItem[0].addSelectionListener(new SelectionAdapter() {
        public void widgetSelected(SelectionEvent e) {
            FileDialog dialog = new FileDialog(sShell, SWT.OPEN);
            dialog.setFilterExtensions(new String[] { "*.doc", "*.txt" });
            String file = dialog.open();
            if (file != null) {

```

```

        openFile = new File(file);
        //打开 OLE 控件
        createOle();
    }
}
});
fileItem[1].addSelectionListener(new SelectionAdapter() {
    public void widgetSelected(SelectionEvent e) {
        //如果打开文件被修改过
        if (clientSite.isDirty()) {
            //创建一个临时文件
            File tempFile = new File(openFile.getAbsolutePath() + ".tmp");
            openFile.renameTo(tempFile);
            //如果保存成功，则删除临时文件，否则恢复到临时文件保存的状态
            if (clientSite.save(openFile, true))
                tempFile.delete();
            else
                tempFile.renameTo(openFile);
        }
    }
});
}
}
}

```

该 OLE 控件的程序只是实现了简单地打开、编辑和保存的功能，另外在 `org.eclipse.swt.ole.win32` 包中还有一些涉及 OLE 控件的类，可以实现其他的一些功能，如果读者有兴趣可以到 MSDN 网站上了解有关 OLE 控件的详细信息。

13.5 Pocket PC 应用

Java 不仅可以应用在计算机中，也可以应用到移动设备如手机或者 PDA 中，同样 SWT 程序也可以应用到移动设备中。如图 13.11 所示为安装了 Microsoft® Pocket PC 操作系统的一个文件选择对话框的程序运行后的效果示意图。

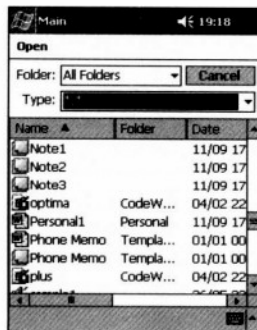


图 13.11 Pocket PC 系统下运行的 SWT 程序

在移动设备中使用的 SWT 程序与之前学习的 SWT 程序编写代码的方式没有什么不同, 例如, 如图 13.11 所示的程序, 代码如下:

```
import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;
public class FileDialogMain {

    public static void main(String[] args) {
        Display display = new Display();
        Shell shell = new Shell(display);
        shell.setText("Main");
        Menu menu = new Menu(shell, SWT.BAR);
        shell.setMenuBar(menu);
        shell.open();
        FileDialog dialog = new FileDialog(shell, SWT.OPEN);
        String name = dialog.open();
        if (name != null)
            shell.setText(name);
        while (!shell.isDisposed()) {
            if (!display.readAndDispatch())
                display.sleep();
        }
        display.dispose();
    }
}
```

但与之前 SWT 程序所不同的是, SWT 程序运行的环境不同, 要在移动设备上运行 SWT 程序, 需要作以下配置:

(1) 确保移动设备已经安装了虚拟机 (VM), 例如 J2ME 的运行环境, 来确保能运行 Java 程序。

(2) 下载并安装 SWT 所需的类库 swt.jar 和与本地系统相关的.dll 文件。获取的方式可以在 Eclipse 网站的下载页面获取。例如, 在 2.2 节中直接获取 SWT 类库的页面中, 选择下载 “Windows CE (ARM PocketPC)” 文件即可获得移动设备支持的 SWT 类库。

因为移动设备的内存和存储的资源有限, 所以与桌面应用的 SWT 类库相比, 移动设备上使用的 SWT 类库大小上少很多。在移动设备上运行的 SWT 类库只包含一些常用的类包如下:

- ☐ org.eclipse.swt.
- ☐ org.eclipse.swt.widgets.
- ☐ org.eclipse.swt.events.
- ☐ org.eclipse.swt.graphics.
- ☐ org.eclipse.swt.layout.

可以说移动设备上的 SWT 类库是简化了的桌面使用的 SWT 类库。

(3) 最后运行编译好的类文件。在移动设备上不可能安装 Eclipse, 所以要使用命令行来运行程序。

当然，移动设备上的应用并不是很广泛，但笔者想要说的是，SWT 不仅可以应用到桌面上，也可以应用到移动设备。如果读者对移动开发有兴趣，可以多进行这方面的实践。

13.6 Web 应用 SWT

SWT 只能用于桌面程序 C/S 结构的开发，那么在网络应用大行其道的今天能不能也将 SWT 程序应用于 Web 开发呢？通过第三方厂商的开发是可以实现的，有一款产品 SmartSWT 便能满足以上要求，读者可以访问 <http://www.smartswt.net/smartswt/index.htm> 来查看该产品的详细信息。当然值得庆幸的是，这也是国人开发的一套产品。

安装了 SmartSWT 服务器和客户端，就可以通过 IE 来访问 SWT 程序，与我们平时浏览网页没什么区别了。如图 13.12 所示为在 IE 中访问的网页显示了 SWT 程序运行的效果图。

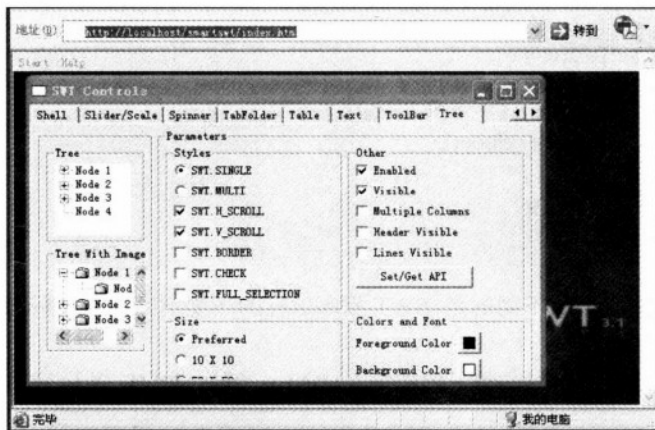


图 13.12 浏览器来访问 SWT 程序

且不说这种方式运行 SWT 的利与弊，但就从应用上来说，确实丰富了 SWT 程序应用的领域。

13.7 本章小结

本章是 SWT 程序的扩展应用，首先介绍了如何使用打印的功能，然后介绍了如何调用使用系统应用程序的办法，之后又讲述了在 SWT 中如何开发 AWT 程序和使用 OLE 程序，最后又介绍了 SWT 除了应用于桌面程序外，也可以应用到移动设备和 Web 中。总之，本章的内容在实际项目中并不常用，读者只需要简单了解一下即可。

至此，有关 SWT 的知识已经讲述完毕了，读者应该对 SWT 有了一个全面的认识。从下章开始，我们就进入到另一个精彩的 JFace 世界了。

第 4 篇



DESIGN

JFace 篇

- 第 14 章 JFace 概述
- 第 15 章 应用程序窗口
- 第 16 章 JFace 对话框
- 第 17 章 向导式对话框
- 第 18 章 首选项
- 第 19 章 MVC 的表格、树和列表
- 第 20 章 JFace 的工具类
- 第 21 章 文本处理

设计知识

PDG

第 14 章 JFace 概述

从本章开始,读者将进入精彩的 JFace 世界。本章是从 SWT 程序到 JFace 程序的过渡,将讲述 JFace 与 SWT 程序的不同之处,快速将读者的思维转变到 JFace 的开发框架中,并且使读者对 JFace 有一个整体的把握,对以后章节学习 JFace 打下一个良好的基础。

14.1 配置 JFace 运行环境

与使用其他 Java 类库一样,运行 JFace 程序也需要导入 JFace 类库。与 SWT 类库不同的是,JFace 的开发包不能从 Eclipse 网站上下载,只能从下载的 Eclipse 中通过 Eclipse 类库来获取。打开 Eclipse 所在的文件夹,打开 plugins 文件夹。例如,笔者将 Eclipse 放置在 F:\javaDev\ 文件夹下,那么打开 F:\javaDev\eclipse\plugins 文件夹,在该文件夹下找到如下的 Jar 包:

- ☐ org.eclipse.jface.text_3.1.2.jar。
- ☐ org.eclipse.jface_3.1.1.jar。
- ☐ org.eclipse.core.runtime_3.1.2.jar。
- ☐ org.eclipse.core.runtime.compatibility_3.1.0.jar。
- ☐ org.eclipse.osgi_3.1.2.jar。
- ☐ org.eclipse.core.commands_3.1.0.jar。

这些包都是运行 JFace 所需要的类包。如果使用 Eclipse 来创建项目,可以使用 Eclipse 内置的导入 JFace 类库的方法,可以自动导入 JFace 类库。在第 3.3.2 节中曾经讲述过的如何导入项目使用的包,就是以导入 JFace 类库为例的。如果此时读者还没有在项目导入 JFace 类库,请参考该小节的内容。导入 JFace 类库后的项目如图 14.1 所示。

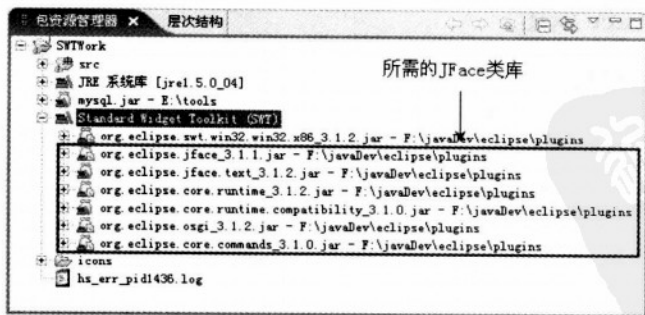


图 14.1 导入 JFace 类库后的项目示意图

注意: 笔者这里用的是 Eclipse 3.1.1, 所以, 由于版本不同包名的版本号可能不同。

导入 JFace 类库后,就配置完了 JFace 运行环境,接下来就要看如何编写一个 JFace 程序了。

14.2 第一个 JFace 程序

首先回顾一下之前学习的第一个 SWT 程序。在 4.1 节中,编写了一个带有按钮的窗口程序。现在看一下如何使用 JFace 来实现同样的功能。如图 14.2 所示为使用 JFace 创建的一个窗口和按钮程序的效果图。

该程序的代码如下:



图 14.2 JFace 程序运行后效果图

HelloJFace.java

```
package com.fengmanfei.ch14;
import org.eclipse.jface.window.ApplicationWindow;
import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Control;
import org.eclipse.swt.widgets.Display;

public class HelloJFace extends ApplicationWindow {

    public HelloJFace() {
        super(null);
    }
    /* (非 Javadoc)
     * @see org.eclipse.jface.window.Window#createContents(org.eclipse.swt.widgets.Composite)
     * 该方法负责创建窗口中的控件
     */
    protected Control createContents(Composite parent) {
        //getShell()方法为父类中的方法,返回窗口的 Shell 对象
        //设置窗口显示的标题
        getShell().setText("JFace");
        //创建一个按钮
        Button button = new Button(parent, SWT.CENTER);
        button.setText("欢迎进入到 JFace 的世界!");
        //调整窗口布局
        parent.pack();
        return parent;
    }

    public static void main(String[] args) {
        //创建窗口对象
        HelloJFace helloJFace = new HelloJFace();
        //设置窗口处于打开状态
        helloJFace.setBlockOnOpen( true );
    }
}
```

```

//打开窗口
helloJFace.open();
//释放 Display 对象
Display.getCurrent().dispose();
}
}

```

虽然该程序与之前使用的 SWT 程序比较起来较复杂,但结构上更加清晰,符合面向对象程序设计(OO)的思路。下面详细讲解一下程序的整体思路。

(1) 首先从 main 方法开始,先创建一个 HelloJFace 对象。HelloJFace 类继承自 ApplicationWindow 类,这个类是封装 Shell 窗口的窗口类。简单地可以理解为,ApplicationWindow 是将 Shell 对象进行了包装,使窗口对象能够被继承地创建,并且封装了一些底层的事件处理。这样做的好处是极大地提高了代码的重用。而在以前使用的 SWT 程序中创建 Shell 窗口的代码几乎是一样的。在程序设计时要尽量避免重复的代码。

(2) HelloJFace 对象创建后,将调用 createContents 方法来创建窗口中的各种控件。这里只需要做的是覆盖父类中的 createContents 方法,在该方法中添加控件内容。其中 parent 对象为父窗口对象,如果要想设置窗口的属性,可以调用父类中 getShell()方法后的窗口对象的引用。之后在窗口上创建各种控件,最后程序返回一个 Control 对象。这个程序中返回的是 parent 面板对象。因为从继承结构上来说 Composite 为 Control 的子类,所以可以返回 parent。另外,若要想获得此返回的 Control 对象,可通过 getContents()获得。

⚠注意:为了区别该方法是否是覆盖父类中的方法,下文的程序代码中凡是覆盖其父类的方法都会添加以下注释:

```

/* (非 Javadoc)
 * @see org.eclipse.jface.window.Window#createContents(org.eclipse.swt.widgets.Composite)
 */
表示该方法是覆盖父类 org.eclipse.jface.window.Window 类中的 createContents 方法。

```

(3) 最后,添加控件后使用 setBlockOnOpen(true)方法和 open()方法打开窗口,类似于 SWT 程序中的 while 的事件循环。最后释放 Display 对象。

如图 14.3 和图 14.4 所示将 SWT 程序运行的过程与 JFace 程序运行的过程作了比较。通过比较读者可更直观地看出 SWT 窗口与 JFace 窗口的不同之处。

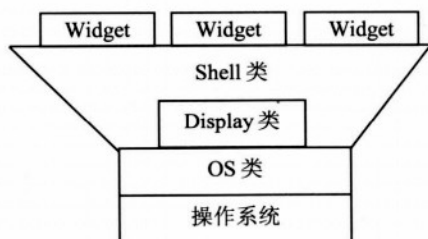


图 14.3 SWT 程序结构图

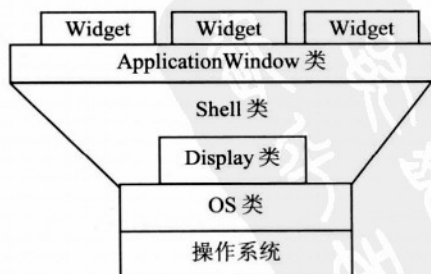


图 14.4 JFace 程序结构图

综上所述, JFace 创建窗口主要是通过继承来实现的。读者在这里可能感受不到继承带来的好处, 随着学习的不断深入, 读者就更能体会到 JFace 使用的方便之处了。

14.3 JFace 框架概述

什么是框架? 在 Web 开发中有许多开发框架, 其中很流行的有 Struts、Hibernate 和 Spring 等。框架是一种设计程序的思想, 使用框架开发大大简化了代码, 并且对以后代码的重用和维护都起到很重要的作用。

对 SWT 而言, JFace 就是 SWT 的框架。JFace 将底层的 SWT 对象进行了封装, 使之更易于重用。当然框架并不是必需的, 如果你有更聪明和更有效的办法实现功能, 当然也可以开发出自己的框架。但笔者认为, 设计软件的思想是, 尽量在前人的基础上进行开发, 如果前人设计的框架不能满足需求时, 再开发自己的框架。因为毕竟在前人的基础上, 会节省许多代码的编写工作。

既然了解了 JFace 与 SWT 的关系, 那么 JFace 框架究竟有什么好处呢? 以下列举了 JFace 对 SWT 改进的几个方面。

- ❑ 应用程序窗口: 通常一个软件都要有一个主窗口, 主窗口一般有菜单栏、工具栏和状态栏, 有了这样的窗口, 就可以轻松地给主窗口添加各种控件, 而不用再编写其他布局之类的代码。
- ❑ 对话框和向导式对话框: 除了主程序窗口, 另外一个常用的就是对话框窗口。对话框窗口一般输入一些信息。另外, JFace 还提供了向导式的对话框, 可以将输入的信息分成几页进行输入。
- ❑ MVC 的树、表格和列表: 在 SWT 中的表格、树等控件, 数据和视图高度耦合, 不利于数据和视图的分离, JFace 改进了这种状况, 使用了 MVC 的设计模式。
- ❑ 首选项: 它也是应用程序中必备的, 通常根据用户的喜好来设定。JFace 提供了创建首选项很简单的方法。
- ❑ 改进了对资源的管理: JFace 将资源集中起来进行管理, 有效地对资源进行释放。
- ❑ 增强了对文本的操作: SWT 中 StyledText 对象已经具有了对文本一定的编辑功能。但 JFace 中文本编辑的功能更加强大, 例如可以折叠代码等。这些功能都可以在 Eclipse 编辑器中找到。Eclipse 的文本编辑功能就建立在 JFace 对文本操作的基础上。
- ❑ JFace 改进了对动作 (Action) 的处理和对后台线程监视等功能。

总之, JFace 框架中的对象都是开发一个应用程序中所常用的。掌握了这些内容, 就可以开发出高效的应用程序。

14.4 JFace 的包结构

JFace 的一些类是针对 Eclipse 平台的, 这些类一般都在 org.eclipse.ui.开头的包中。另外,

独立于 Eclipse 平台的一般都在 org.eclipse.jface.开头的包中。这里着重学习 org.eclipse.jface.包中的类。

通过 Eclipse 自带的帮助系统，可以查阅相关的 JFace API 参考。在 Eclipse 工作平台菜单上选择“帮助”|“帮助内容”|“平台插件开发者指南”|“参考”|“API 参考”命令，弹出如图 14.5 所示的 JFace API 参考窗口。

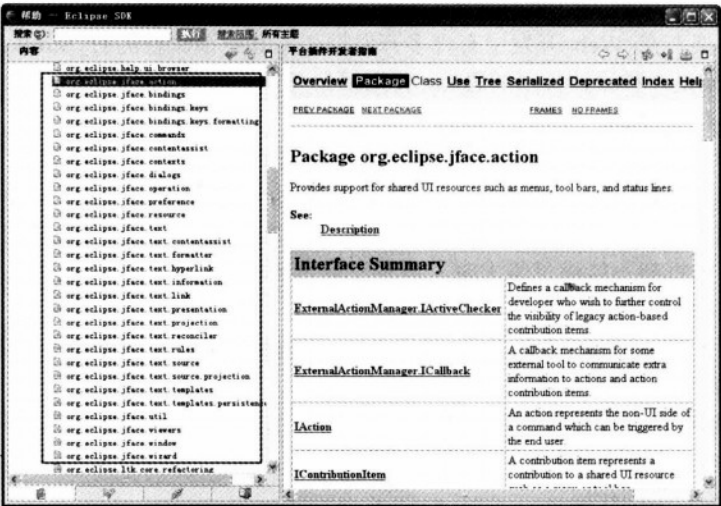


图 14.5 JFace API 参考

表 14.1 列举了 JFace 包中各类的主要功能和用途。

表 14.1 JFace 包结构概述

包 名	包 描 述
org.eclipse.jface.action	该包主要是封装了一些菜单，工具栏上的按钮动作类，这样对同一个动作可以在不同的控件上使用
org.eclipse.jface.bindings	这些包主要绑定了一些快捷键等
org.eclipse.jface.bindings.keys	
org.eclipse.jface.bindings.keys.formatting	
org.eclipse.jface.commands	该包主要提供了命令类，主要供 JFace 内部使用
org.eclipse.jface.contentassist	该包提供了对一些控件的内容辅助的功能
org.eclipse.jface.contexts	JFace 的上下文，一般供 JFace 内部管理使用
org.eclipse.jface.dialogs	JFace 的各种对话框类
org.eclipse.jface.operation	主要涉及对线程的控制
org.eclipse.jface.preference	设置首选项有关的类
org.eclipse.jface.resource	涉及有关资源管理的类
org.eclipse.jface.text	涉及文本编辑的一些类
org.eclipse.jface.text.*	
org.eclipse.jface.util	工具类，一般为 JFace 内部调用

续表

包 名	包 描 述
org.eclipse.jface.viewers	MVC 模式设计的表格、树和列表等
org.eclipse.jface.window	窗口类和应用程序窗口
org.eclipse.jface.wizard	有关涉及对话框窗口向导的类

14.5 本章小结

通过对本章的学习，您对 JFace 的框架应该有了一个大概的认识。JFace 的编程更多地采用了继承的方式，与之前学习的 SWT 程序的结构有很大不同，希望读者能尽快从 SWT 的编程方式转变过来，习惯 JFace 的编程方式。



第 15 章 应用程序窗口

一个应用程序的运行通常是以创建一个主窗口开始的，所以本章着重讲述如何使用 JFace 的窗口类。JFace 中有关窗口的接口和类都在 org.eclipse.jface.window 包下。本章将重点学习该包中的 Window 类和 ApplicationWindow 类。

15.1 JFace 的窗口类（Window 类）

ApplicationWindow 类在第 14 章的第一个 JFace 程序中已经接触过，它是一个窗口类，其实它是继承自 Window 类。Window 类是一个抽象类，ApplicationWindow 是 Window 类的一个实现，当然还有另一个实现就是 Dialog 类（对话框类），将在第 16 章详细讲述。从图 15.1 所示的类结构继承关系图中，读者可以看出它们之间的继承关系。

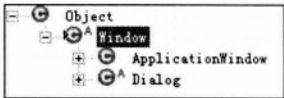


图 15.1 窗口类的继承关系图

简单地说，Window 类是对 Shell 对象的第一层封装，它提供了一个抽象的窗口，包括对窗口底层的事件处理和创建窗口的一些方法。对于这个类，读者只需要了解它的一些常用方法即可，在适当的时候在继承时可覆盖其方法。

表 15.1 列举了该类的一些方法及其说明。

表 15.1 Window 类的常用方法及说明

方 法 名	描 述
boolean canHandleShellCloseEvent()	是否调用关闭窗口事件，默认值为 true，将会调用 handleShellCloseEvent()方法
boolean close()	关闭窗口，释放资源，从窗口管理器中移出该窗口
void configureShell(Shell newShell)	设置 Shell 窗口的一些属性，通常为设置标题、图标、布局等
Layout getLayout()	获得窗口的布局
void constrainShellSize()	限制窗口的大小不超过显示的边界
void create()	初始化窗口和窗口的控件，将调用 createContents 方法和 createShell 方法
Control createContents(Composite parent)	初始化控件，通常需覆盖该方法
final Shell createShell()	初始化主窗口，注意该方法为 final 型，不能被覆盖
Control getContents()	获得 createContents 返回的控件对象
static Image getDefaultImage()	获得为窗口默认设置的图标
static Image[] getDefaultImages()	获得新打开窗口图标的数组对象

续表

方 法 名	描 述
<code>Point getInitialLocation(Point initialSize)</code>	获得窗口初始状态下的坐标位置
<code>Point getInitialSize()</code>	获得窗口初始状态下的大小
<code>Shell getParentShell()</code>	获得该窗口的父窗口对象
<code>int getReturnCode()</code>	返回窗口的状态, 如果处于打开状态, 则返回 <code>Window.OK</code> 常量; 如果关闭, 则返回 <code>Window.CANCEL</code> 常量
<code>Shell getShell()</code>	返回创建的该窗口对象
<code>ShellListener getShellListener()</code>	返回窗口的监听器
<code>int getShellStyle()</code>	返回窗口的样式, 默认为 <code>SWT.CLOSE SWT.MIN SWT.MAX SWT.RESIZE</code> 。若要设置窗口的样式, 可调用 <code>setShellStyle</code> 方法设置
<code>WindowManager getWindowManager()</code>	返回窗口管理器对象
<code>void handleFontChange(PropertyChangeEvent event)</code>	处理字体改变时的方法, 通常需要覆盖该方法来实现
<code>void handleShellCloseEvent()</code>	关闭窗口触发的事件
<code>void initializeBounds()</code>	初始化窗口的位置和大小, 调用 <code>getInitialSize</code> 和 <code>getInitialLocation</code> 方法
<code>int open()</code>	打开窗口
<code>setBlockOnOpen(boolean shouldBlock)</code>	阻止窗口关闭
<code>static void setDefaultImage(Image image)</code>	设置默认的窗口图标
<code>static void setDefaultImages(Image[] images)</code>	设置默认窗口图标数组对象
<code>void setParentShell(Shell newParentShell)</code>	设置该窗口的父窗口
<code>void setReturnCode(int code)</code>	设置窗口的状态, 与 <code>getReturnCode</code> 对应
<code>Rectangle getConstrainedShellBounds(Rectangle preferredSize)</code>	获得窗口所设定的最佳位置和大小
<code>void setShellStyle(int newShellStyle)</code>	设置窗口的样式
<code>void setWindowManager(WindowManager manager)</code>	为该窗口设置窗口管理器
<code>static void setExceptionHandler(Window.IExceptionHandler handler)</code>	抛出异常时处理的方法
<code>static void setDefaultModalParent(IShellProvider provider)</code>	设置默认的窗口模式
<code>static int getDefaultOrientation()</code>	获得默认窗口控件的方向, 获得的值为 <code>SWT.RIGHT_TO_LEFT</code> 、 <code>SWT.LEFT_TO_RIGHT</code> 和 <code>SWT.NONE</code>
<code>static void setDefaultOrientation(int defaultOrientation)</code>	设置默认窗口控件的方向与 <code>getDefaultOrientation</code> 方法相对应

在使用 `Window` 类编写代码的过程中, 并不需要全部覆盖父类的方法, 通常继承时只需要覆盖其中的一些方法。通常被子类覆盖的方法有:

- ❑ close: 关闭窗口可以释放一些资源。
- ❑ configureShell: 在窗口打开前设置一个窗口的属性。
- ❑ createContents: 在窗口打开前创建窗口的内容。
- ❑ getInitialSize: 设置窗口初始大小。
- ❑ getInitialLocation: 设置窗口的初始位置。
- ❑ getShellListener: 设置窗口的事件监听器。
- ❑ getToolTipText: 设置提示。
- ❑ handleFontChange: 处理字体改变的方法，通常可以用于改变窗口的字体同时也改变控件的字体。
- ❑ handleShellCloseEvent: 处理关闭窗口时的事件处理。

15.2 应用程序窗口 ApplicationWindow 类

ApplicationWindow 类是应用程序的窗口类，该类除了 Window 类常用的方法外，还添加了设置菜单栏、工具栏、状态栏等方法。这些控件一般都是应用程序的主窗口需要的。另外该类实现了 IRunnableContext 接口，可以使用线程来创建一个任务，并且将任务的状态显示到窗口的状态栏上，所以使用 ApplicationWindow 类可以轻易地创建出主窗口。如图 15.2 所示，即为一个带有菜单栏、工具栏和状态栏的应用程序的窗口类。

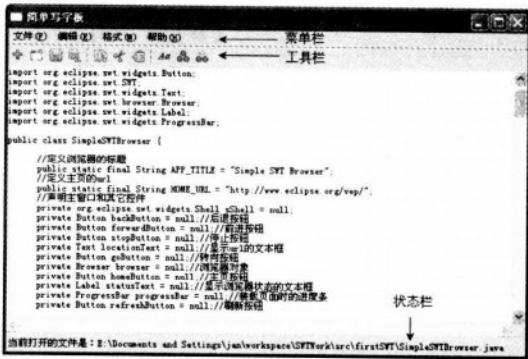


图 15.2 一个典型的应用程序主窗口

首先看一下 ApplicationWindow 类常用的方法，然后再详细讲述如何来创建所需要的菜单栏、工具栏和状态栏等控件。表 15.2 列举了 ApplicationWindow 类中的一些方法。

表 15.2 ApplicationWindow 类的常用方法

方 法 名	描 述
void addMenuBar()	添加菜单栏，通常要覆盖 createMenuManager 方法来创建菜单的控件
MenuManager createMenuManager()	创建菜单栏的管理器对象
MenuManager getMenuBarManager()	获得 createMenuManager 方法创建的菜单管理器对象

续表

方 法 名	描 述
<code>void addToolBar(int style)</code>	添加工具栏，通常要覆盖 <code>createToolBarManager</code> 方法来创建工具栏中的按钮
<code>ToolBarManager createToolBarManager(int style)</code>	创建工具栏的管理器对象
<code>ToolBarManager getToolBarManager()</code>	获得 <code>createToolBarManager</code> 方法创建的工具栏管理器对象
<code>createToolBarControl(Composite parent)</code>	创建工具栏控件
<code>Control getToolBarControl()</code>	获得 <code>createToolBarControl</code> 创建的工具栏
<code>void addCoolBar(int style)</code>	添加可拖动的工具栏。通常要覆盖 <code>createCoolBarManager</code> 方法创建可拖动工具栏的控件
<code>CoolBarManager createCoolBarManager(int style)</code>	创建可拖动工具栏的管理器对象
<code>CoolBarManager getCoolBarManager()</code>	获得 <code>createCoolBarManager</code> 方法创建的可拖动工具栏管理器对象
<code>Control createCoolBarControl(Composite composite)</code>	创建可拖动工具栏的控件
<code>Control getCoolBarControl()</code>	获得 <code>createCoolBarControl</code> 方法返回的控件对象
<code>void addStatusLine()</code>	添加状态栏，通常要覆盖 <code>createStatusLineManager</code> 方法来显示状态栏的内容
<code>StatusLineManager createStatusLineManager()</code>	创建状态栏管理器
<code>StatusLineManager getStatusLineManager()</code>	获得 <code>createStatusLineManager</code> 方法创建的状态栏管理器对象
<code>createStatusLine(Shell shell)</code>	创建状态栏
<code>run(boolean fork, boolean cancelable, IRunnableWith Progress runnable)</code>	可以开辟一个线程，并显示进度
<code>void setStatus(String message)</code>	设置状态栏显示的字符
<code>boolean showTopSeperator()</code>	是否显示菜单栏与主窗口的分割线

15.3 带有菜单栏的主程序窗口

了解了一些常用的方法之后，下面来看一下如何创建主窗口程序所需要的各种控件了。当要在主窗口显示菜单栏时，首先在构造方法中调用 `this.addMenuBar()` 方法，此方法将调用父类 `createMenuManager()` 方法，此时将菜单项加入到该方法中。

15.3.1 简单写字板程序示例

以下是在图 15.2 所示的简单剪贴板程序中添加了菜单栏的程序代码，请读者注意

createMenuManager 方法中的代码。

MainWindow.java

```
package com.fengmanfei.ch15;

import org.eclipse.jface.action.*;
import org.eclipse.jface.window.ApplicationWindow;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.*;
import org.eclipse.swt.widgets.*;
import com.fengmanfei.ch15.actions.*;

public class MainWindow extends ApplicationWindow {

    private NewAction newAction; //新建菜单项
    private OpenAction openAction; //打开菜单项
    private SaveAction saveAction; //保存菜单项
    private SaveAsAction saveAsAction; //另存为菜单项
    private ExitAction exitAction; //退出菜单项
    private CopyAction copyAction; //复制菜单项
    private CutAction cutAction; //剪切菜单项
    private PasteAction pasteAction; //粘贴菜单项
    private HelpAction helpAction; //帮助菜单项
    private FileManager manager; //文件管理器
    private Text content; //文本框
    private static MainWindow app; //主程序窗口
    private MainWindow() {
        super(null);
        app = this;
        manager = new FileManager();
        //初始化按钮动作
        newAction = new NewAction();
        openAction = new OpenAction();
        saveAction = new SaveAction();
        saveAsAction = new SaveAsAction();
        exitAction = new ExitAction();
        copyAction = new CopyAction();
        cutAction = new CutAction();
        pasteAction = new PasteAction();
        helpAction = new HelpAction();

        this.addMenuBar(); //添加菜单栏
        this.addToolBar(SWT.FLAT); //添加工具栏
        this.addStatusLine(); //添加状态栏
    }

    /**
     * @return 返回 app
     */
}
```

```

public static MainWindow getApp() {
    return app;
}
/*
 * (非 Javadoc)
 *
 * @see org.eclipse.jface.window.ApplicationWindow#configureShell (org.eclipse.swt.
widgets.Shell)
 */
protected void configureShell(Shell shell) {
    // TODO 自动生成方法存根
    super.configureShell(shell);
    shell.setText("简单写字板");
    shell.setMaximized(true); // 设置最大化
}

/* (非 Javadoc)
 * @see org.eclipse.jface.window.Window#createContents(org.eclipse.swt.widgets. Composite)
 */
protected Control createContents(Composite parent) {
    content = new Text(parent, SWT.MULTI | SWT.V_SCROLL | SWT.H_SCROLL);
    content.addModifyListener( new ModifyListener(){
        public void modifyText(ModifyEvent e) {
            manager.setDirty( true );
        }
    });
    return parent;
}

/*
 * (非 Javadoc)
 *
 * @see org.eclipse.jface.window.ApplicationWindow#createMenuManager()
 */
protected MenuManager createMenuManager() {
    MenuManager menuBar = new MenuManager(); // 创建的菜单栏对象

    MenuManager fileMenu = new MenuManager("文件(&F)"); // 文件菜单项
    MenuManager editMenu = new MenuManager("编辑(&E)"); // 编辑菜单项
    MenuManager formatMenu = new MenuManager("格式(&F)"); // 格式菜单项
    MenuManager helpMenu = new MenuManager("帮助(&H)"); // 帮助菜单项

    // 将菜单项添加到主菜单中
    menuBar.add(fileMenu);
    menuBar.add(editMenu);
    menuBar.add( formatMenu );
    menuBar.add(helpMenu);

    // 文件菜单项

```

```
fileMenu.add(newAction);
fileMenu.add(openAction);
fileMenu.add(new Separator());
fileMenu.add(saveAction);
fileMenu.add(saveAsAction);
fileMenu.add(new Separator());
fileMenu.add(exitAction);

editMenu.add(copyAction);
editMenu.add(cutAction);
editMenu.add(pasteAction);

formatMenu.add( new FormatAction( FormatAction.TYPE_FONT));
formatMenu.add( new FormatAction( FormatAction.TYPE_BGCOLOR));
formatMenu.add( new FormatAction( FormatAction.TYPE_FORECOLOR));

helpMenu.add(helpAction);
return menuBar;

}

public static void main(String[] args) {
    MainWindow main = new MainWindow();
    main.setBlockOnOpen(true);
    main.open();
    Display.getCurrent().dispose();
}

/**
 * @return 返回 content
 */
public Text getContent() {
    return content;
}

/**
 * @return 返回 manager
 */
public FileManager getManager() {
    return manager;
}

/**
 * @param manager 要设置的 manager
 */
public void setManager(FileManager manager) {
    this.manager = manager;
}
}
```

15.3.2 添加菜单栏的基本步骤

首先要在构造方法中调用父类的 `this.addMenuBar()` 方法，然后再覆盖父类的 `createMenuManager()` 方法，在该方法中添加具体的菜单内容。简单的代码如下：

```
public class MainWindow extends ApplicationWindow {
    private MainWindow() {
        super(null);
        this.addMenuBar();//添加菜单栏
    }
    protected MenuManager createMenuManager() {
        MenuManager menuBar = new MenuManager();
        //添加具体的菜单项
        return menuBar;
    }
}
```

15.3.3 创建菜单项

JFace 将创建菜单的控件 `Menu` 和 `MenuItem` 包装成 `MenuManager` 对象，通过 `MenuManager` 可以轻松地创建窗口的菜单项。为了更好地说明菜单是如何创建的，只创建几个简单的菜单项。代码如下：

```
MenuManager menuBar = new MenuManager();// 创建主菜单对象
MenuManager fileMenu = new MenuManager("文件(&F)"); // 文件菜单项
MenuManager editMenu = new MenuManager("编辑(&E)"); // 编辑菜单项
// 将菜单项添加到主菜单中
menuBar.add(fileMenu);
menuBar.add(editMenu);
//为文件菜单项添加
fileMenu.add(new Action());
fileMenu.add(openAction);
fileMenu.add(new Separator());//添加分割线
fileMenu.add(exitAction);
```

代码运行后如图 15.3 所示。每个菜单项添加到主菜单使用 `add` 方法添加。另外菜单项是由 `add` 方法添加一个 `Action` 对象实现的，有关 `Action` 的处理方法将在 15.3.4 节详细讲解。

如果想要添加二级子菜单，只要再创建一个 `MenuManager` 对象，然后添加到菜单项中即可。如在上述代码中添加以下代码即可创建一个子菜单。

```
MenuManager formatMenu = new MenuManager("格式(&F)"); // 格式菜单项
//格式菜单的菜单项
formatMenu.add( new FormatAction( FormatAction.TYPE_FONT));
formatMenu.add( new FormatAction( FormatAction.TYPE_BGCOLOR));
formatMenu.add( new FormatAction( FormatAction.TYPE_FORECOLOR));
//将该菜单添加到文件菜单项中
fileMenu.add( formatMenu );
```


程序运行后效果如图 15.4 所示。

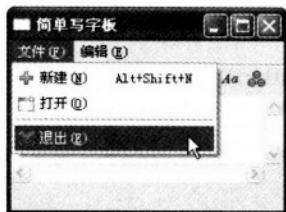


图 15.3 程序运行后的示意图

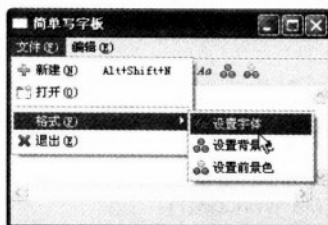


图 15.4 二级菜单

总之，要创建一个菜单需要创建一个 MenuManager 对象，然后通过 add 方法添加其菜单项，这些菜单项可以是菜单也可以是单独一个可触发事件的按钮。

15.3.4 菜单项的事件处理

在添加菜单项的 add 方法中，使用的是一个 Action 对象。例如，添加“退出”按钮的菜单项的代码如下：

```
ExitAction exitAction = new ExitAction();
fileMenu.add(exitAction);
```

其中 ExitAction 类继承自 Action 类，表示一个用户单击的动作事件。下面具体看一下 ExitAction 类的代码：

ExitAction.java

```
package com.fengmanfei.ch15.actions;

import org.eclipse.jface.action.Action;
import org.eclipse.jface.resource.ImageDescriptor;
import com.fengmanfei.ch15.*;

public class ExitAction extends Action {
    public ExitAction() {
        super();
        setText("&退出(E)");//设置菜单项的文本
        setToolTipText("退出系统");//设置提示
        setImageDescriptor(ImageDescriptor.createFromFile(NewAction.class, "icons\\exit.gif"));// 设置图标文件
    }
    /*
     * (非 Javadoc)
     *
     * @see org.eclipse.jface.action.Action#run()
     * 需覆盖此方法，用户单击后，将执行 run()方法中的事件
     */
    public void run() {
```

```
MainWindow.getApp().close();//关闭窗口
    }
}
```

要使菜单项具有处理事件的功能，需要覆盖父类的 `run()` 方法，然后在该方法中添加处理事件的方法，例如这里的退出事件处理很简单，单击退出后即关闭窗口。

其实这样将处理的事件与触发事件的 UI 控件分离开来有一个好处就是，对于一个关闭窗口操作，可以出现在菜单栏中，也可以出现在工具栏中，有时也可能出现在上下文菜单中，这样就可以直接调用同一个处理事件的方法了。有关更复杂的一些事件的处理将在 15.7 节详细讲述。

15.4 带有工具栏的主程序窗口

创建工具栏的方法与创建菜单栏的方法类似，首先需在构造方法中调用 `addToolBar` 方法，然后再覆盖父类中的 `createToolBarManager` 方法。创建菜单栏的代码如下：

```
public class MainWindow extends ApplicationWindow {
    private MainWindow() {
        super(null);
        this.addToolBar(SWT.FLAT);//添加工具栏
    }

    /* (非 Javadoc)
     * @see org.eclipse.jface.window.ApplicationWindow#createToolBarManager(int)
     */
    protected ToolBarManager createToolBarManager(int style) {
        ToolBarManager toolBar = new ToolBarManager( style );
        toolBar.add(new Action());
        toolBar.add(openAction);
        toolBar.add(saveAction);
        toolBar.add(saveAsAction);
        toolBar.add(new Separator());//工具栏的分割线
        toolBar.add(copyAction);
        toolBar.add(cutAction);
        toolBar.add(pasteAction);
        toolBar.add(new Separator());
        toolBar.add( new FormatAction( FormatAction.TYPE_FONT));
        toolBar.add( new FormatAction( FormatAction.TYPE_BGCOLOR));
        toolBar.add( new FormatAction( FormatAction.TYPE_FORECOLOR));
        return toolBar;
    }
}
```

同样，创建工具栏按钮是由 `ToolBarManager` 对象创建的，该对象与 `MenuManager` 对象类似。从这里就可以看出使用 `Action` 对象来处理事件的优势了，只需要将 `Action` 对象通过

add 方法添加到工具栏中, 就可以与菜单栏中的退出按钮具有相同的功能了。

另外, 如果想创建可拖动的工具栏, 方法与创建普通工具栏的方法类似, 需在构造方法中调用 addCoolBar 方法, 然后覆盖 createCoolBarManager 方法即可。可移动的工具栏中的按钮是通过 CoolBarManager 来创建的。

15.5 带有状态栏的主程序窗口

状态栏也是与添加菜单和工具栏的方法相同, 在构造方法中调用 addStatusLine 方法, 然后覆盖父类 create StatusLineManager 方法, 另外状态栏所对应的管理器对象为 StatusLineManager。

如果想在状态栏上显示一些状态信息, 可以使用 StatusLineManager 对象的 setMessage (String message) 方法设置。状态栏还有一个重要的作用就是显示后台线程的进度情况, 特别是在打开比较大的文件时, 需要在后台开辟一个线程来处理打开文件的操作。下面以打开文件操作为例, 看一下如何将后台的进度实现在状态栏中。

如图 15.5 所示为打开一个较大文件时在状态栏中显示的打开状态信息的示意图。



图 15.5 显示打开进度的状态栏

打开菜单项所对应的 Action 对象为 OpenAction 对象。以下为 OpenAction 类的代码:

OpenAction.java

```
package com.fengmanfei.ch15.actions;

import java.lang.reflect.InvocationTargetException;
import org.eclipse.core.runtime.IProgressMonitor;
import org.eclipse.jface.action.Action;
import org.eclipse.jface.operation.IRunnableWithProgress;
import org.eclipse.jface.operation.ModalContext;
import org.eclipse.jface.resource.ImageDescriptor;
import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.FileDialog;

import com.fengmanfei.ch15.FileManager;
import com.fengmanfei.ch15.MainWindow;

public class OpenAction extends Action {

    public OpenAction() {
        super();
        setText("打开(&O)");
        setToolTipText("打开文件");
    }
}
```

```

setImageDescriptor(ImageDescriptor.createFromFile(NewAction.class,"icons\\open.gif"));
}

/*
 * (非 Javadoc)
 *
 * @see org.eclipse.jface.action.Action#run()
 */
public void run() {
    //打开一个文件对话框, 选择一个文件
    FileDialog dialog = new FileDialog(MainWindow.getApp().getShell(), SWT.OPEN);
    dialog.setFilterExtensions(new String[] { "*.java", "**.*" });
    final String name = dialog.open();
    if ((name == null) || (name.length() == 0))
        return;
    //创建一个 FileManager 对象, 该对象封装了从文件中读取字符串的方法
    final FileManager fileManager = MainWindow.getApp().getManager();
    try {
        //创建一个线程打开文件
        ModalContext.run(new IRunnableWithProgress() {
            //线程运行的主体
            public void run(IPrimaryProgressMonitor progressMonitor) {
                progressMonitor.beginTask("打开文件", IProgressMonitor.UNKNOWN);
                fileManager.load(name);
                //为了模拟一个较大的文件, 让该线程休息 1 秒
                /*取消该注释运行可以看到进度情况
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                */
                progressMonitor.done();
            }
        }, true, MainWindow.getApp().getStatusLineManager().getProgressMonitor(),
        MainWindow.getApp().getShell().getDisplay());
    } catch (InvocationTargetException e) {
        e.printStackTrace();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    //装载后将字符显示到文本框中
    MainWindow.getApp().getContent().setText(fileManager.getContent());
    //设置状态栏显示的是打开的文件目录
    MainWindow.getApp().getStatusLineManager().setMessage("当前打开的文件是: "+name);
}
}

```

实现该程序需要注意以下几方面:

(1) 创建线程的方法是利用 JFace 提供的创建线程的工具类 `ModalContext` 类。在 `org.eclipse.jface.operation` 包中, 通过该类的静态方法创建一个后台的线程。

```
run(IRunnableWithProgress operation,boolean fork, IProgressMonitor monitor,Display display)
```

下面解释一下每个参数所代表的意义。

- ❑ `operation`: 是线程体, 也就是线程所要执行的代码。
- ❑ `fork`: 是否在创建线程时另一个执行, `true` 为开辟一个新线程, 否则在该线程中执行。
- ❑ `monitor`: 进度显示的控件, 本程序中以状态栏显示进度情况, 因为状态栏控件 `org.eclipse.jface.action.StatusLine` 类实现了 `IProgressMonitor` 接口。
- ❑ `display`: UI 线程对象。

(2) 文件打开后, 通过 `StatusLineManager` 对象的 `setMessage` 方法设置状态栏中所显示的信息。

(3) 最后, 来看一下 `FileManager` 类, 这个类的作用是对文件进行管理, 例如装载一个文件和保存一个文件等。该类的代码如下:

FileManager.java

```
package com.fengmanfei.ch15;
import java.io.*;
public class FileManager {

    private String fileName;//文件名
    private boolean dirty = false;//文件是否被修改过
    private String content;//文件的字符串
    public FileManager() {
    }
    public void load(String name) { //加载一个文件
        final String textString;
        try {
            File file = new File(name);
            FileInputStream stream = new FileInputStream(file.getPath());
            Reader in = new BufferedReader(new InputStreamReader(stream));
            char[] readBuffer = new char[2048];
            StringBuffer buffer = new StringBuffer((int) file.length());
            int n;
            while ((n = in.read(readBuffer)) > 0) {
                buffer.append(readBuffer, 0, n);
            }
            textString = buffer.toString();
            stream.close();
        } catch (FileNotFoundException e) { //如果读文件出错, 将错误信息显示到状态栏
            MainWindow.getApp().getStatusLineManager().setMessage("文件未找到:" +
            fileName);
            return;
        } catch (IOException e) {
            MainWindow.getApp().getStatusLineManager().setMessage("读文件出错:" +
```



```

fileName);
        return;
    }
    content = textString;//将文件的字符保存到 content 中
    this.fileName = name;
}

public void save(String name) { //保存一个文件
    final String textString = content;
    try {
        File file = new File(name);
        FileOutputStream stream = new FileOutputStream(file.getPath());
        Writer out = new OutputStreamWriter( stream );
        out.write( textString );
        out.flush();
        stream.close();
    } catch (FileNotFoundException e) {
        MainWindow.getApp().getStatusLineManager().setMessage("文件未找到:" +
        fileName);
        return;
    } catch (IOException e) {
        MainWindow.getApp().getStatusLineManager().setMessage("保存文件出错:" +
        fileName);
        return;
    }
}

/**
 * @return 返回 content
 */
public String getContent() {
    return content;
}

/**
 * @param content 要设置的 content
 */
public void setContent(String content) {
    this.content = content;
}

/**
 * @return 返回 dirty
 */
public boolean isDirty() {
    return dirty;
}

/**
 * @param dirty 要设置的 dirty
 */
public void setDirty(boolean dirty) {
    this.dirty = dirty;
}

```

```
}  
/**  
 * @return 返回 fileName  
 */  
public String getFileName() {  
    return fileName;  
}  
/**  
 * @param fileName 要设置的 fileName  
 */  
public void setFileName(String fileName) {  
    this.fileName = fileName;  
}  
}
```

15.6 其他处理事件的方法

最后, 为了完善该简单写字板的程序, 也使读者进一步对 Action 的使用有一定的了解, 以下显示了各个按钮所执行的操作。

15.6.1 “新建”操作

“新建”按钮的具体代码如下:

NewAction.java

```
package com.fengmanfei.ch15.actions;  
  
import org.eclipse.jface.action.Action;  
import org.eclipse.jface.resource.ImageDescriptor;  
import org.eclipse.swt.SWT;  
import com.fengmanfei.ch15.FileManager;  
import com.fengmanfei.ch15.MainWindow;  
  
public class NewAction extends Action {  
  
    public NewAction() {  
        super();  
        setText("新建(&N)");  
        this.setAccelerator(SWT.ALT + SWT.SHIFT + 'N');  
        setToolTipText("新建");  
        setImageDescriptor(ImageDescriptor.createFromFile(NewAction.class, "icons\\new.gif"));  
    }  
    /* (非 Javadoc)  
    * @see org.eclipse.jface.action.Action#run()  
    */  
}
```

```

public void run() {
    MainWindow.getApp().getContent().setText(" ");
    MainWindow.getApp().setManager( new FileManager());
}
}

```

15.6.2 “保存”操作

“保存”按钮的具体代码如下：

SaveAction.java

```

package com.fengmanfei.ch15.actions;

import org.eclipse.jface.action.Action;
import org.eclipse.jface.resource.ImageDescriptor;
import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.FileDialog;
import org.eclipse.swt.widgets.MessageBox;

import com.fengmanfei.ch15.FileManager;
import com.fengmanfei.ch15.MainWindow;

public class SaveAction extends Action{
    public SaveAction() {
        super();
        setText("&S");
        setToolTipText("保存文件");

        setImageDescriptor( ImageDescriptor.createFromFile(NewAction.class,"icons\\save.gif"));
    }

    /*
     * (非 Javadoc)
     *
     * @see org.eclipse.jface.action.Action#run()
     */
    public void run() {
        final FileManager fileManager = MainWindow.getApp().getManager();
        //如果文件没有修改过，则不保存
        if (!fileManager.isDirty())
            return ;
        //如果是新建的文件，首先要选择保存的路径
        if ( fileManager.getFileName() == null ){
            FileDialog saveDialog = new FileDialog( MainWindow.getApp(). getShell(), SWT.
SAVE);
            saveDialog.setText("请选择所要保存的文件");
            saveDialog.setFilterPath("F:\\");

```

```

        saveDialog.setFilterExtensions(new String[] { "*.java", "**.*" });
        String saveFile = saveDialog.open();
        if ( saveFile != null )
        {
            fileManager.setFileName( saveFile );
            fileManager.setContent( MainWindow.getApp().getContent().getText() );
            fileManager.save( fileManager.getFileName());
        }
        fileManager.setDirty( false );
        return ;
    }
    //如果是打开的文件，弹出对话框确认保存
    if (fileManager.getFileName()!= null){
        MessageBox box = new MessageBox( MainWindow.getApp().getShell() ,SWT.
ICON_QUESTION|SWT.YES|SWT.NO);
        box.setText("保存");
        box.setMessage("您确定要保存文件吗? ");
        int choice = box.open();
        if (choice==SWT.NO)
            return;
        fileManager.setContent( MainWindow.getApp().getContent().getText() );
        fileManager.save( fileManager.getFileName());
        fileManager.setDirty( false );
        return ;
    }
}
}

```

15.6.3 “另存为” 操作

“另存为” 按钮的具体代码如下：

SaveAsAction.java

```

package com.fengmanfei.ch15.actions;

import org.eclipse.jface.action.Action;
import org.eclipse.jface.resource.ImageDescriptor;
import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.FileDialog;
import com.fengmanfei.ch15.FileManager;
import com.fengmanfei.ch15.MainWindow;

public class SaveAsAction extends Action {
    public SaveAsAction() {
        super();
        setText("&A");
        setToolTipText("另存为");
        setImageDescriptor(ImageDescriptor.createFromFile(NewAction.class, "icons\\saveas.gif"));
    }
}

```



```

    }
    /*
     * (非 Javadoc)
     *
     * @see org.eclipse.jface.action.Action#run()
     */
    public void run() {
        final FileManager fileManager = MainWindow.getApp().getManager();
        //弹出保存对话框
        FileDialog saveDialog = new FileDialog( MainWindow.getApp().getShell(), SWT.SAVE);
        saveDialog.setText("请选择所要保存的文件");
        saveDialog.setFilterPath("F:\\");
        saveDialog.setFilterExtensions(new String[] { "*.java", "*.*" });
        String saveFile = saveDialog.open();
        if ( saveFile != null )
        {
            fileManager.setFileName( saveFile );
            fileManager.setContent( MainWindow.getApp().getContent().getText() );
            fileManager.save( fileManager.getFileName());//保存文件
        }
        fileManager.setDirty( false );
        return ;
    }
}

```

15.6.4 “复制”、“剪切”和“粘贴”操作

“复制”、“剪切”和“粘贴”按钮的代码类似，不同的是在 `run()` 方法中调用文本框对象的 `copy()`、`cut()` 和 `paste()` 方法。以下显示的是“剪切”按钮的事件处理的代码：

CutAction.java

```

package com.fengmanfei.ch15.actions;

import org.eclipse.jface.action.Action;
import org.eclipse.jface.resource.ImageDescriptor;
import org.eclipse.swt.SWT;

import com.fengmanfei.ch15.MainWindow;

public class CutAction extends Action {
    public CutAction() {
        super();
        setText("剪切(&C)");
        setToolTipText("剪切");
        setAccelerator( SWT.CTRL + 'X');
        setImageDescriptor(ImageDescriptor.createFromFile(NewAction.class, "icons\\cut.gif"));
    }
}

```



```

/* (非 Javadoc)
 * @see org.eclipse.jface.action.Action#run()
 */
public void run() {
    MainWindow.getApp().getContent().cut();
}
}

```

设置前景色、背景色和字体的按钮是通过一个 Action 来实现的，在创建 Action 对象时指定一个 type 类型，然后根据不同的类型来构造不同的事件。该类的代码如下：

FormatAction.java

```

package com.fengmanfei.ch15.actions;

import org.eclipse.jface.action.Action;
import org.eclipse.jface.resource.ImageDescriptor;
import org.eclipse.swt.graphics.*;
import org.eclipse.swt.widgets.*;

import com.fengmanfei.ch15.MainWindow;

public class FormatAction extends Action {
    public static final String TYPE_FORECOLOR = "FORECOLOR";
    public static final String TYPE_BGCOLOR = "BGCOLOR";
    public static final String TYPE_FONT = "FONT";
    private String formatType;// 通过不同的类型来构造不同的处理事件
    public FormatAction(String type) {
        super();
        this.formatType = type;
        initAction();
    }

    private void initAction() {
        if (formatType.equals(TYPE_FONT)) {
            this.setText("设置字体");
            this.setToolTipText("设置字体");
            setImageDescriptor(ImageDescriptor.createFromFile(NewAction.class,
"icons\\font.gif"));
        } else if (formatType.equals(TYPE_FORECOLOR)) {
            this.setText("设置前景色");
            this.setToolTipText("设置前景色");
            setImageDescriptor(ImageDescriptor.createFromFile(NewAction.class,
"icons\\foreColor.gif"));
        } else {
            this.setText("设置背景色");
            this.setToolTipText("设置背景色");
            setImageDescriptor(ImageDescriptor.createFromFile(NewAction.class,
"icons\\bgColor.gif"));
        }
    }
}

```

```

    }

}

/*
 * (非 Javadoc)
 * @see org.eclipse.jface.action.Action#run()
 */
public void run() {
    Text content = MainWindow.getApp().getContent();
    if (formatType.equals(TYPE_FONT)) {
        FontDialog fontDialog = new FontDialog(MainWindow.getApp().getShell());
        fontDialog.setFontList(content.getFont().getFontData());
        FontData fontData = fontDialog.open();
        if (fontData != null) {
            Font font = new Font(MainWindow.getApp().getShell().getDisplay(), fontData);
            content.setFont(font);
        }
    } else if (formatType.equals(TYPE_FORECOLOR)) {
        ColorDialog colorDialog = new ColorDialog(MainWindow.getApp().getShell());
        colorDialog.setRGB(content.getForeground().getRGB());
        RGB rgb = colorDialog.open();
        if (rgb != null) {
            Color foregroundColor = new Color(MainWindow.getApp().getShell().getDisplay(), rgb);
            content.setForeground(foregroundColor);
            foregroundColor.dispose();
        }
    } else {
        ColorDialog colorDialog = new ColorDialog(MainWindow.getApp().getShell());
        colorDialog.setRGB(content.getBackground().getRGB());
        RGB rgb = colorDialog.open();
        if (rgb != null) {
            Color bgColor = new Color(MainWindow.getApp().getShell().getDisplay(), rgb);
            content.setBackground(bgColor);
            bgColor.dispose();
        }
    }
}
}
}

```

最后，提醒读者的是：使用 Action 的方法处理事件时，仅限于菜单栏、工具栏和按钮的单击事件的处理，若想添加其他的事件，还要使用 addXXXListener 的方法实现。

15.7 本章小结

本章以一个简单的写字板程序演示了如何利用 JFace 的应用程序窗口类，快速地创建添加菜单栏、工具栏和状态栏控件。使用框架可以减少代码的编写量，但却减少程序的灵活性。所以当 JFace 框架不能满足用户所要实现的功能时，要开发出自己的一套框架也是完全有必要的。



第 16 章 JFace 对话框

本章将详细介绍 JFace 框架中的对话框。在 SWT 的学习中已经介绍过一些简单的对话框，但 JFace 中的对话框要复杂得多，包括可以直接显示信息的对话框、输入信息的对话框和带有进度条的对话框等。

16.1 JFace 对话框概述

对话框也是常见的控件，JFace 已经封装好了一些常用的对话框类，这样通过使用这些对话框类，可以方便地创建一个对话框。JFace 中的对话框类都在 `org.eclipse.jface.dialogs` 包中，如图 16.1 所示为 JFace 对话框类的继承关系示意图。

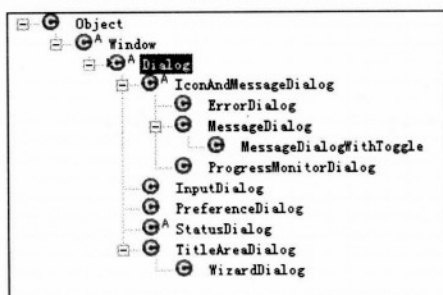


图 16.1 JFace 对话框继承关系示意图

这些对话框类主要有以下功能：

- ❑ **ErrorDialog**：可根据错误级别来显示错误信息，一般用于 Eclipse 工作台。
- ❑ **MessageDialog**：可显示提示信息的对话框，类似于 SWT 的对话框，但比 SWT 功能强大，是最常用的对话框。
- ❑ **MessageDialogWithToggle**：一般在保存首选项所设置的值时显示是否保存的提示信息。该对话框将在第 18 章中详细讲解。
- ❑ **ProgressMonitorDialog**：可以显示后台线程进度的对话框。
- ❑ **InputDialog**：用于输入信息的对话框，同时可以验证用户的输入，并可以将验证的信息显示在对话框中。
- ❑ **PreferenceDialog**：专门为设置首选项所使用的对话框。该对话框也将在第 18 章详细讲解。
- ❑ **TitleAreaDialog**：带有标题、图标和描述信息的对话框。
- ❑ **WizardDialog**：向导式对话框，用于将一个操作分成几页来显示的对话框。将在第

17 章详细讲解。

16.2 信息提示对话框 (MessageDialog)

在 SWT 中, 使用 `MessageBox` 对话框可以显示提示的消息, 但是 `MessageBox` 对话框不能自定义显示的按钮。而 `MessageDialog` 不仅可以自定义显示的按钮, 还能够自定义显示区域的内容。如图 16.2 所示为一个带有自定义按钮的信息提示对话框。

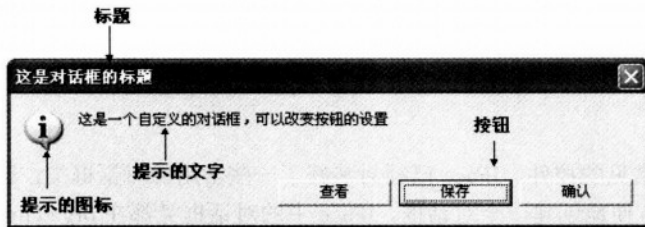


图 16.2 带有自定义按钮的信息提示对话框

创建这样的对话框所使用的构造方法是:

```
MessageDialog(Shell parentShell, String dialogTitle,  
              Image dialogTitleImage, String dialogMessage, int dialogImageType,  
              String[] dialogButtonLabels, int defaultIndex)
```

其中各参数的说明如下:

- ❑ `parentShell`: 对话框所属的 `Shell` 窗口。
- ❑ `dialogTitle`: 对话框显示的标题。
- ❑ `dialogTitleImage`: 对话框的图标, 为 `null` 则不显示图标。
- ❑ `dialogMessage`: 对话框提示的文字。
- ❑ `dialogImageType`: 对话框提示的图标, 还可以选择使用 `MessageDialog.ERROR`、`MessageDialog.QUESTION` 和 `MessageDialog.WARNING` 常量。
- ❑ `dialogButtonLabels`: 按顺序显示在对话框上的按钮。
- ❑ `defaultIndex`: 打开对话框时, 默认选中的按钮索引值。

16.2.1 创建信息提示对话框

下面写一个测试窗口, 该窗口用一个文本框作为输出的控制台, 然后单击不同的按钮弹出不同的对话框, 并将返回值输出到控制台中。该测试窗口运行后, 单击“自定义对话框”按钮, 即可打开如图 16.2 所示的对话框。该测试窗口的代码如下:

MessageDialogTest.java

```
package com.fengmanfei.ch16;
```



```

import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.jface.window.*;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.*;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.widgets.*;

public class MessageDialogTest extends ApplicationWindow {

    public MessageDialogTest() {
        super(null);
    }

    protected void configureShell(Shell shell) {
        super.configureShell(shell);
        shell.setSize(550,300);
        shell.setText("消息对话框示例");
    }

    protected Control createContents(Composite parent) {
        //窗口中的区域由一个文本框作为消息输出台和几个按钮分别打开不同的对话框
        Composite composite = new Composite( parent ,SWT.NONE);
        GridLayout gridLayout = new GridLayout (6,false);
        composite.setLayout( gridLayout );
        final Text console = new Text ( composite ,SWT.NONE|SWT.READ_ ONLY|SWT.
V_SCROLL);
        GridData data = new GridData(GridData.FILL_BOTH);
        data.horizontalSpan=6;
        console.setLayoutData( data);
        Button customMessageDig = new Button( composite ,SWT.NONE);
        customMessageDig.setText("自定义对话框");
        customMessageDig.addSelectionListener( new SelectionAdapter(){
            public void widgetSelected(SelectionEvent e) {
                MessageDialog dialog = new MessageDialog(Display.getCurrent(). GetActive
Shell(),//Shell 窗口
                    "这是对话框的标题",//标题
                    null,//对话框的图标, 为 null 则不显示图标
                    "这是一个自定义的对话框, 可以改变按钮的设置",//对话框中的提示
                    MessageDialog.INFORMATION,//提示信息的图标
                    new String[]{"查看","保存","确认"},//显示 3 个按钮
                    1);//默认选择的按钮的索引值, 这里为 1, 表示默认选中第二个按钮,
也就是"保存"按钮

                int i = dialog.open();//返回值为按钮
                console.append("\n 自定义对话框, 返回按钮的索引值"+i);
            }
        });
        return parent;
    }
}

```

```

    }
    public static void main(String[] args) {
        MessageDialogTest test = new MessageDialogTest();
        test.setBlockOnOpen( true );
        test.open();
        Display.getCurrent().dispose();
    }
}

```

该测试窗口运行后如图 16.3 所示。

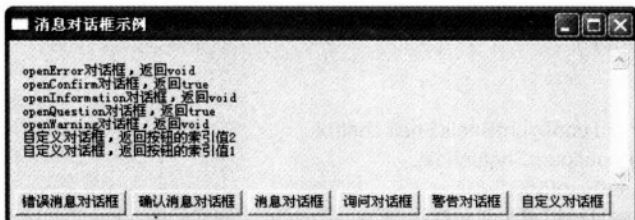


图 16.3 测试窗口程序运行后示意图

另外，为了方便建立消息对话框，MessageDialog 类提供了 5 个静态的方法，可以直接创建预置的对话框。

16.2.2 错误消息对话框

打开错误消息对话框使用的静态方法如下：

```
static void openError(Shell parent, String title, String message)
```

带有一个“确定”按钮，并且显示错误图标，返回值为 void。如图 16.4 所示为错误消息对话框。

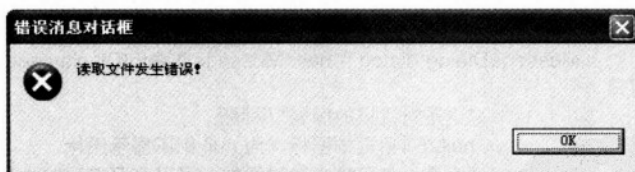


图 16.4 错误消息对话框

打开这样的一个对话框的代码如下：

```

Button openError = new Button( composite ,SWT.NONE);
openError.setText("错误消息对话框");
openError.addSelectionListener( new SelectionAdapter(){
    public void widgetSelected(SelectionEvent e) {
        MessageDialog.openError(Display.getCurrent().getActiveShell(),"错误消息对话框",
            "读取文件发生错误! ");
        console.append("\n openError 对话框, 返回 void");
    }
}

```

```
    }  
});
```

16.2.3 确认消息对话框

打开确认消息对话框使用的静态方法如下：

```
static void openConfirm (Shell parent, String title, String message)
```

带有“确定”按钮和“取消”按钮，并且显示询问的图标，如果单击“确定”按钮，则返回 true，否则返回 false。如图 16.5 所示为确认消息对话框。

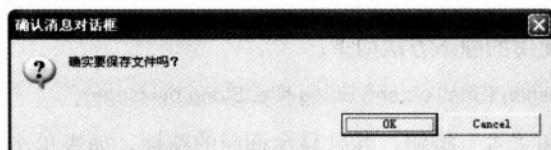


图 16.5 确认消息对话框

打开这样的一个对话框的代码如下：

```
Button openConfirm = new Button( composite ,SWT.NONE);  
openConfirm.setText("确认消息对话框");  
openConfirm.addSelectionListener( new SelectionAdapter(){  
    public void widgetSelected(SelectionEvent e) {  
        boolean b = MessageDialog.openConfirm(Display.getCurrent().getActiveShell(),  
            "确认消息对话框",  
            "确实要保存文件吗? ");  
        console.append("\n openConfirm 对话框, 返回"+b);  
    }  
});
```

16.2.4 消息对话框

打开消息对话框使用的静态方法如下：

```
static void openInformation (Shell parent, String title, String message)
```

带有一个“确定”按钮，并且显示消息的图标，返回 void。如图 16.6 所示为消息对话框。

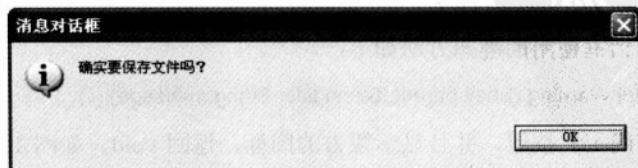


图 16.6 消息对话框

打开这样的一个对话框的代码如下：

```
Button openInformation = new Button( composite ,SWT.NONE);
openInformation.setText("消息对话框");
openInformation.addSelectionListener( new SelectionAdapter(){
    public void widgetSelected(SelectionEvent e) {
        MessageDialog.openInformation(Display.getCurrent().getActiveShell(),"消息对话框","确实要保存文件吗? ");
        console.append("\n openInformation 对话框, 返回 void");
    }
});
```

16.2.5 询问对话框

打开询问对话框使用的静态方法如下:

```
static void openQuestion (Shell parent, String title, String message)
```

带有“是”按钮和“否”按钮,并且显示询问的图标,如果单击“是”按钮,则返回 true, 否则返回 false。如图 16.7 所示为询问对话框。

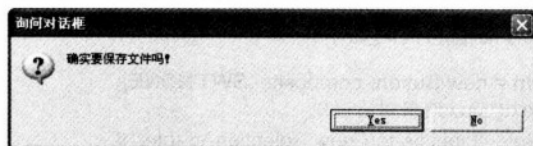


图 16.7 询问对话框

打开这样的一个对话框的代码如下:

```
Button openQuestion = new Button( composite ,SWT.NONE);
openQuestion.setText("询问对话框");
openQuestion.addSelectionListener( new SelectionAdapter(){
    public void widgetSelected(SelectionEvent e) {
        boolean b = MessageDialog.openQuestion(Display.getCurrent().getActiveShell(),"询问对话框","确实要保存文件吗! ");
        console.append("\n openQuestion 对话框, 返回"+b);
    }
});
```

16.2.6 警告对话框

打开警告对话框使用的静态方法如下:

```
static void openWarning (Shell parent, String title, String message)
```

带有一个“确定”按钮,并且显示警告的图标,返回 void。如图 16.8 所示为警告询问对话框。

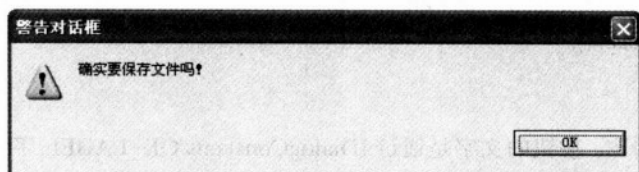


图 16.8 警告对话框

打开这样的一个对话框的代码如下：

```
Button openWarning = new Button( composite ,SWT.NONE);
openWarning.setText("警告对话框");
openWarning.addSelectionListener( new SelectionAdapter(){
    public void widgetSelected(SelectionEvent e) {
        MessageDialog.openWarning(Display.getCurrent().getActiveShell(),"警告对话框","确实
        要保存文件!");
        console.append("\n openWarning 对话框, 返回 void");
    }
});
```

16.2.7 JFace 的本地化

细心的读者可能已经发现，在显示 JFace 对话框中的中文出现了问题。为什么 JFace 对话框中显示的按钮为英文的，而在 SWT 中同样的对话框却显示的是中文呢？如图 16.9 所示为 SWT 对话框和 JFace 的对话框的比较示意图。

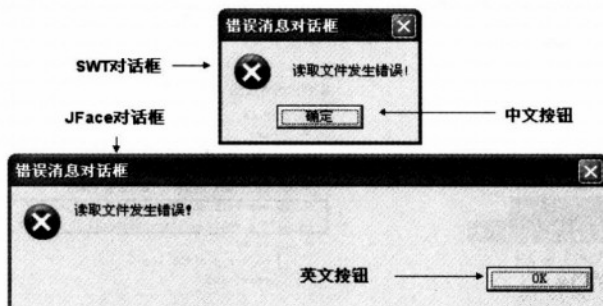


图 16.9 SWT 对话框和 JFace 对话框中的中文问题

1. 本地化产生的原因

这是因为 SWT 中的对话框是调用本地操作系统的对话框，它显示的按钮是根据本地操作系统的语言设定的。而 JFace 中的对话框是通过 SWT 中的 Shell 窗口封装来的，调用的是 SWT 中的 Button 控件对象，查看 MessageDialog 类的源代码可以发现设置按钮文字是通过字符串常量来设定的。例如，以下为 MessageDialog 类中打开错误消息对话框的代码：

```
public static void openError(Shell parent, String title, String message) {
    MessageDialog dialog = new MessageDialog(parent, title, null, message, ERROR, new
```



```
String[] { IDialogConstants.OK_LABEL }, 0);
    dialog.open();
    return;
}
```

这里可以看出,按钮的文字是通过 IDialogConstants.OK_LABEL 字符常量来设定的,而该常量的值在 IDialogConstants 类中的值为:

```
public String OK_LABEL = JFaceResources.getString("ok");
```

然后在 JFaceResources 类中可以发现,这个字符是根据 org.eclipse.jface 包中 messages*.properties 文件所指定的 ResourceBundle 文件来获取的,也就是获取的本地的语言包。

```
private static final ResourceBundle bundle = ResourceBundle.getBundle ("org.eclipse. jface.
messages");
```

当然,现在所运行的 JFace 环境中并没有这个语言包,所以不能显示中文,只能以默认的英文来显示。所以,要使 JFace 的对话框本地化,要找到语言包然后导入到项目工程中即可。

2. 安装 JFace 语言包

如何获取 JFace 的语言包呢?其实很简单,因为之前在安装 Eclipse 时,已经安装了 Eclipse 的多国语言包,此时在 Eclipse 的安装目录的 plugins 文件夹下,例如笔者这里使用的路径是“F:\javaDev\eclipse\plugins”,然后找到如图 16.10 所示的包。当然这里只使用 org.eclipse.jface.nl1_3.1.1.jar 包就可以解决本地化按钮的问题,在此为了以后运行的 JFace 程序全部为本地化,所以同时使用两个包。

找到这两个语言包后,按照 3.2.2 节中导入项目使用包的方法,将这两个包添加到项目工程中来,如图 16.11 所示。

```
org.eclipse.jdt_3.1.0.jar
org.eclipse.jface.nl1_3.1.1.jar
org.eclipse.jface.text.nl1_3.1.1.jar
org.eclipse.jface.text_3.1.2.jar
org.eclipse.jface_3.1.1.jar
```

图 16.10 JFace 的语言包

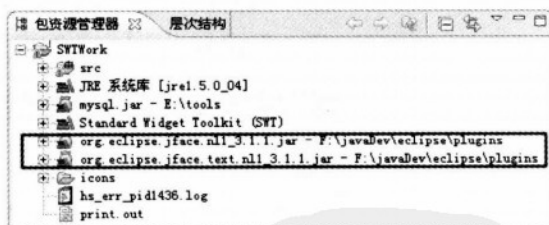


图 16.11 导入 JFace 语言包的项目示意图

配置好上述工作后,现在重新运行一下刚才的程序,是不是 JFace 对话框中的按钮都变成中文了呢?而且将这两个包导入后,其他有关涉及多语言的文本也都为中文了。

16.3 输入对话框 (InputDialog)

输入对话框 (InputDialog 类) 可以提示用户输入一段文本,并且也判断输入字符的有

效性, 当输入错误时将错误信息返回给用户。如图 16.12 所示为一个提示用户输入电子邮件的输入对话框。

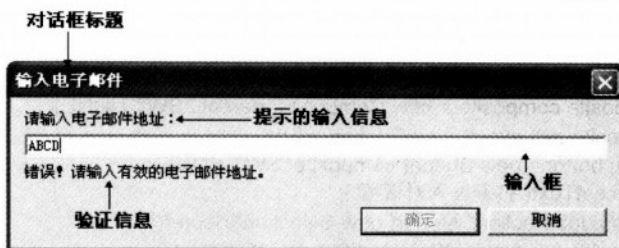


图 16.12 输入对话框

创建输入对话框的构造方法是:

```
public InputDialog(Shell parentShell,String dialogTitle,
String dialogMessage,String initialValue,IInputValidator validator)
```

其中各参数所代表的意义如下所示:

- ❑ parentShell: 对话框所属的 Shell 窗口。
- ❑ dialogTitle: 对话框的标题。
- ❑ dialogMessage: 提示用户输入信息的文本。
- ❑ initialValue: 打开对话框时, 输入文本中默认显示的字符。
- ❑ validator: 验证用户输入有效性的对象, null 表示不需要验证。要实现验证功能的类必须实现 IInputValidator 接口。

16.3.1 创建输入对话框

创建如图 16.12 所示的对话框的程序代码如下:

InputDialogTest.java

```
package com.fengmanfei.ch16;

import org.eclipse.jface.dialogs.InputDialog;
import org.eclipse.jface.window.ApplicationWindow;
import org.eclipse.jface.window.Window;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.*;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.*;

public class InputDialogTest extends ApplicationWindow{

    public InputDialogTest() {
        super(null);
```

```

    }
    protected void configureShell(Shell shell) {
        shell.setText("输入对话框示例");
    }

    protected Control createContents(Composite parent) {
        Composite composite = new Composite( parent , SWT.NONE);
        composite.setLayout( new GridLayout());
        Button button= new Button( composite ,SWT.NONE);
        button.setText("打开输入对话框");
        button.addSelectionListener( new SelectionAdapter(){
            public void widgetSelected(SelectionEvent e) {
                InputDialog inputDialog = new InputDialog(Display.getCurrent(). GetActive
Shell(),
                    "输入电子邮件",//对话框的标题
                    "请输入电子邮件地址: ",//对话框的提示信息
                    "ABC@hotmail.com",//输入框中默认值
                    new EmailValidator()//验证输入字符的有效性
                );
                int r = inputDialog.open();//打开窗口
                if (r==Window.OK)//如果输入有效, 则输出输入的值
                    System.out.println(inputDialog.getValue());
                else
                    System.out.println("取消");
            }
        });
        return parent;
    }
    public static void main(String[] args) {
        InputDialogTest test = new InputDialogTest();
        test.setBlockOnOpen( true );
        test.open();
        Display.getCurrent().dispose();
    }
}

```

该程序运行后的效果如图 16.13 所示。单击“打开输入对话框”按钮，即可弹出输入对话框。

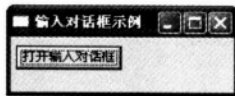


图 16.13 程序运行后示意图

16.3.2 创建输入文本的验证类

另外，为了验证用户输入的文本是否符合电子邮件的格式，还需要写一个验证类 `EmailValidator`，该类需实现 `IInputValidator` 接口。`EmailValidator` 类具体实现的代码如下：

EmailValidator.java

```

package com.fengmanfei.ch16;
import org.eclipse.jface.dialogs.IInputValidator;

public class EmailValidator implements IInputValidator{
    public String isValid(String newText) {
        //如果输入的字符串中不含@字符，输入无效
        if (newText.indexOf("@")==-1)
            return "错误！请输入有效的电子邮件地址。";
        return null;
    }
}

```

16.4 带有提示信息的对话框（TitleAreaDialog）

带有提示信息的对话框（TitleAreaDialog 类）可在标题上显示提示的信息，提示信息类别的不同，将显示不同的图标。如图 16.14 所示为一个带提示信息的对话框。

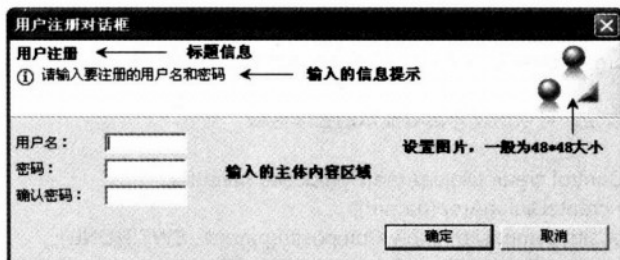


图 16.14 带有提示信息的对话框示意图

带有提示信息的对话框可以根据用户当前的输入提示错误的消息，创建这样的对话框需使用继承的方式，然后覆盖父类的方法。如图 16.14 所示的程序代码如下：

InputPasswordDialog.java

```

package com.fengmanfei.ch16;

import org.eclipse.jface.dialogs.IMessageProvider;
import org.eclipse.jface.dialogs.TitleAreaDialog;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.FocusAdapter;
import org.eclipse.swt.events.FocusEvent;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.*;
import com.fengmanfei.util.ImageFactory;

public class InputPasswordDialog extends TitleAreaDialog{

```

```

private Text userName;
private Text password;
private Text confirmPassword;
public final String DEFAULT_INFO="请输入要注册的用户名和密码";
public InputPasswordDialog(Shell parentShell) {
    super(parentShell);
}
/* (非 Javadoc)
 * @see org.eclipse.jface.dialogs.TitleAreaDialog#createContents(org.eclipse.swt.widgets.
Composite)
 */
protected Control createContents(Composite parent) {
    super.createContents(parent);
    this.getShell().setText("用户注册对话框");//设置对话框标题栏
    this.setTitle("用户注册");//设置标题信息
    this.setMessage("请输入要注册的用户名和密码");//设置初始化对话框的提示信息
    //设置右侧的图片，一般为 48*48 大小
    this.setTitleImage( ImageFactory.loadImage( Display.getCurrent(), ImageFactory.
SAMPLE_ICON));
    return parent;
}

/* (非 Javadoc)
 * @see org.eclipse.jface.dialogs.TitleAreaDialog#createDialogArea(org.eclipse.swt.widgets.
Composite)
 * 覆盖该方法，可以创建对话框显示的主体区域
 */
protected Control createDialogArea(Composite parent) {
    super.createDialogArea(parent);
    Composite composite = new Composite(parent , SWT.NONE);
    composite.setLayout( new GridLayout(2,true));
    new Label( composite , SWT.NONE).setText("用户名: ");
    userName = new Text(composite ,SWT.BORDER);
    userName.addListener( new FocusAdapter(){
        //当用户名文本框失去焦点时，判断是否有效
        public void focusLost(FocusEvent e) {
            checkValid();
        }
    });
    new Label( composite , SWT.NONE).setText("密码: ");
    password = new Text(composite ,SWT.BORDER);
    password.setEchoChar("");
    new Label( composite , SWT.NONE).setText("确认密码: ");
    confirmPassword = new Text(composite ,SWT.BORDER);
    confirmPassword.setEchoChar("");
    confirmPassword.addListener( new FocusAdapter(){
        //当确认密码文本框失去焦点时，判断是否有效
        public void focusLost(FocusEvent e) {
            checkValid();
        }
    });
}

```



```

    }
    });
    return parent;
}
//判断是否输入有效, 并提示用户
protected void checkValid() {
    if (!password.getText().equals(confirmPassword.getText()))
        setMessage("确认密码不一致, 请重新输入!", IMessageProvider.WARNING);
    else if (userName.getText().equals(" "))
        setMessage("用户名不能为空!", IMessageProvider.ERROR);
    else
        setMessage(DEFAULT_INFO);
}
}

```

程序运行后, 当“用户名”和“确认密码”文本框失去焦点时都会判断有效性, 并显示不同的提示信息。如图 16.15 所示为在不同的错误下所提示的消息不同。

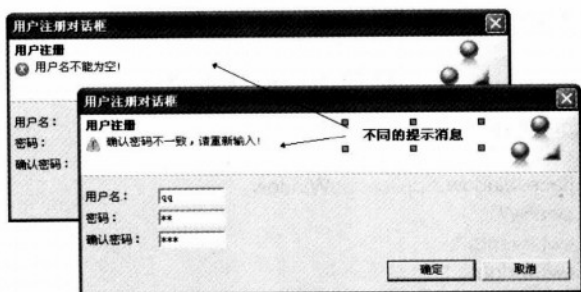


图 16.15 不同的提示信息

在使用 `TitleAreaDialog` 类时, 应注意以下几个问题:

- ❑ 创建窗口所显示的内容要通过继承 `TitleAreaDialog` 类的方式, 而不能直接创建。例如, 运行以下代码将会抛出异常:

```




TitleAreaDialog dialog = new TitleAreaDialog( shell );
dialog.setMessage("这是显示的信息");//错误!
dialog.open();

```

- ❑ 当继承 `TitleAreaDialog` 类时, 通常要调用父类的以下几个方法:
 - ◆ `Control createContents(Composite parent)`: 该方法创建一个顶级容器来放置显示消息的部分和输入区域的部分。
 - ◆ `Control createDialogArea(Composite parent)`: 创建对话框输入区域的控件, 一般都要覆盖此方法。
- ❑ 另外, `TitleAreaDialog` 类还有一些常用的方法可供继承的类调用:
 - ◆ `setErrorMessage(String newErrorMessage)`: 设置错误的提示信息。
 - ◆ `setMessage(String newMessage)`: 设置显示的提示信息。
 - ◆ `setMessage(String newMessage, int newType)`: 可显示不同的图标和提示信息。

- ◆ setTitle(String newTitle): 设置提示信息标题。
 - ◆ setTitleAreaColor(RGB color): 设置标题区域的颜色。
 - ◆ setTitleImage(Image newTitleImage): 设置右侧的图片。
- 在使用 setMessage(String newMessage, int newType)方法显示消息时，可根据 newType 参数选择使用不同的图标。表 16.1 列出了不同样式常量所表示的图标。

表 16.1 信息类型所对应的图标

样 式 常 量	图标示意图
IMessageProvider.ERROR	
IMessageProvider.WARNING	
IMessageProvider.INFORMATION	
IMessageProvider.NONE	不显示任何图标

- 最后，要运行此对话框，还需创建一个窗口来调用该对话框，单击按钮调用此对话框。测试该对话框程序代码如下：

InputPWDDialogTest.java

```
package com.fengmanfei.ch16;

import org.eclipse.jface.window.ApplicationWindow;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.*;
import org.eclipse.swt.widgets.*;

public class InputPWDDialogTest extends ApplicationWindow{
    public InputPWDDialogTest () {
        super(null);
    }
    protected Control createContents(Composite parent) {
        Button button = new Button( parent ,SWT.NONE);
        button.setText("打开输入对话框");
        button.addSelectionListener( new SelectionAdapter(){
            public void widgetSelected(SelectionEvent e) {
                //调用该对话框
                InputPasswordDialog dialog = new InputPasswordDialog(Display.getCurrent().
getActiveShell());
                dialog.open();
            }
        });
        return parent;
    }
    public static void main(String[] args) {
        InputPWDDialogTest test = new InputPWDDialogTest ();
        test.setBlockOnOpen( true );
        test.open();
    }
}
```

```

        Display.getCurrent().dispose();
    }
}

```

16.5 错误提示对话框 (ErrorDialog)

在程序运行时，难免会遇到抛出异常的时候，所以在运行期间有效的错误处理机制能够及时地告知用户错误发生的原因，以便使系统更好地运行。虽然使用 `MessageDialog.openError()` 方法也可以弹出错误消息的对话框，但这种错误对话框的功能有限，而 `ErrorDialog` 错误对话框能够根据错误的级别来显示错误信息，这样设计的好处是，当要在一个对话框中显示多个抛出的错误时，可以根据每个错误的级别来显示。如图 16.16 所示为一个错误提示对话框。

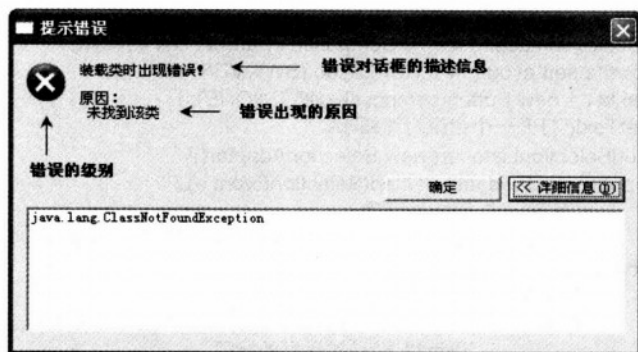


图 16.16 错误提示对话框

16.5.1 创建错误提示对话框

创建错误提示消息的对话框使用的构造方法如下：

```

ErrorDialog(Shell parentShell, String dialogTitle, String message,
            IStatus status, int displayMask)

```

下面解释一下每个参数所代表的意义：

- ❑ `parentShell`: 对话框所属的 Shell 窗口。
- ❑ `dialogTitle`: 对话框的标题。
- ❑ `message`: 对话框的描述信息。
- ❑ `status`: 该对话框中可显示的错误状态对象，`IStatus` 对象将在 16.5.2 节讲述。
- ❑ `displayMask`: 过滤错误的级别，可使用的常量有 `Status.INFO`、`IStatus.ERROR`、`IStatus.WARNING`、`IStatus.OK` 等。如果想同时显示多种状态的错误，可以使用“|”分割。

下面来看一下如何实现如图 16.16 所示的错误对话框程序。程序运行后，单击“打开一个错误对话框”按钮，即可出现如图 16.16 所示的对话框。该程序的代码如下：

AlertDialogTest.java

```

package com.fengmanfei.ch16;
import org.eclipse.core.runtime.IStatus;
import org.eclipse.core.runtime.Status;
import org.eclipse.jface.dialogs.AlertDialog;
import org.eclipse.jface.window.ApplicationWindow;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.*;
import org.eclipse.swt.layout.RowLayout;
import org.eclipse.swt.widgets.*;

public class AlertDialogTest extends ApplicationWindow{
    final String dummyPlugin = "plugin id";
    public AlertDialogTest() {
        super(null);
    }
    protected Control createContents(Composite parent) {
        Composite composite = new Composite ( parent , SWT.NONE);
        composite.setLayout( new RowLayout(SWT.VERTICAL));
        Button bt1 = new Button(composite ,SWT.NONE);
        bt1.setText("打开一个错误对话框");
        bt1.addSelectionListener( new SelectionAdapter(){
            public void widgetSelected(SelectionEvent e) {
                //创建一个 Status 对象
                Status status = new Status(IStatus.ERROR, dummyPlugin, 1, "未找到该类",
new ClassNotFoundException());
                //创建错误提示对话框
                AlertDialog dlg = new AlertDialog(Display.getCurrent().getActiveShell(),
                    "提示错误", //对话框的标题
                    "装载类时出现错误！ ",//对话框的描述信息
                    status, //Status 对象
                    IStatus.ERROR);//只显示级别为 IStatus.ERROR 的错误

                dlg.open();
            }
        });
        return parent;
    }
}

public static void main(String[] args) {
    AlertDialogTest test = new AlertDialogTest();
    test.setBlockOnOpen( true );
    test.open();
    Display.getCurrent().dispose();
}
}

```

另外，为了方便创建错误提示对话框，AlertDialog 提供了静态方法可以直接打开错误对话框。例如，以上创建 AlertDialog 对象的方法改为如下所示情况可以达到同样的效果：

```

AlertDialog.openError(Display.getCurrent().getActiveShell(),
    "提示错误", //对话框的标题

```

```
"装载类时出现错误!", //对话框的描述信息
status, //Status 对象
IStatus.ERROR);
```

16.5.2 使用错误状态对象

在创建 `ErrorDialog` 对话框时, 需要使用 `IStatus` 类。该类是一个接口, 实现该接口的类有 `Status` 和 `MultiStatus` 类。`Status` 可以保存一个错误的状态信息, 而 `MultiStatus` 可以同时保存多个错误状态的信息。例如, 以上的程序中使用的就是 `Status` 对象。该类的构造方法是:

```
public Status(int severity, String pluginId, int code, String message, Throwable exception)
```

其中各参数的意义如下:

- ☐ `severity`: 错误的级别, 有 `Status.INFO`、`IStatus.ERROR`、`IStatus.WARNING`、`IStatus.OK` 和 `Status.CANCEL` 等。
- ☐ `pluginId`: 主要用于 Eclipse 平台中识别插件的唯一标识 ID, 如果不是在 Eclipse 平台下使用, 可以随意设置字符。
- ☐ `code`: 也是与 Eclipse 平台相关的参数。表示 Eclipse 插件状态代码, 如果不是在 Eclipse 平台中使用, 可以随便设置一个整型的数值即可。
- ☐ `message`: 出现该错误时, 提示给用户的信息。
- ☐ `exception`: 异常出现的类型。

例如, 创建一个携带警告信息的 `Status` 对象的代码如下:

```
Status status = new Status(IStatus.WARNING, "plugin_id", 1, "未找到该类", new ClassNotFoundException());
```

16.5.3 同时显示多个错误信息

在实际的程序运行中, 一般情况下不可能只出现一个错误, 而更多的是同时有多个错误出现。这时要想将多个错误信息同时显示给用户就需要使用 `MultiStatus` 对象。`MultiStatus` 对象可以同时存储多个错误对象状态。使用的构造方法是:

```
public MultiStatus(String pluginId, int code, IStatus[] newChildren,
String message, Throwable exception)
```

其中 `newChildren` 表示该错误对象中所包含的多个错误对象。

下面的代码显示的是如何使用显示多个错误状态的对话框。

```
IStatus[] statuses = new IStatus[4];
statuses[0] = new Status(IStatus.INFO, dummyPlugin, IStatus.OK, "未找到类错误", new
ClassNotFoundException());
statuses[1] = new Status(IStatus.ERROR, dummyPlugin, IStatus.OK, "未找到文件错误", new
FileNotFoundException());
statuses[2] = new Status(IStatus.WARNING, dummyPlugin, IStatus.OK, "运行错误", new
RuntimeException());
```



```
statuses[3] = new Status(IStatus.WARNING, dummyPlugin, IStatus.OK, "数据库查询错误",
new SQLException());
MultiStatus multiStatus = new MultiStatus(dummyPlugin, IStatus.OK, statuses, "运行期间错误 ",
new Exception());
ErrorDialog dlg =
new ErrorDialog(Display.getCurrent().getActiveShell(),
    "提示错误", //对话框的标题
    "运行 JFace 期间发生的错误", //对话框的描述信息
    multiStatus, //Status 对象
    IStatus.WARNING | IStatus.ERROR); // 显示级别为 IStatus.WARNING 和 IStatus.
ERROR 的错误
dlg.open();
```

运行以上代码后，所创建的错误显示对话框将显示所有 IStatus.WARNING 和 IStatus.ERROR 级别的错误消息，虽然 MultiStatus 对象中也有 IStatus.INFO 级别的消息，但是被过滤掉了，如图 16.17 所示。

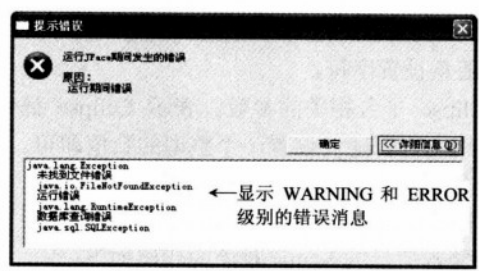


图 16.17 同时显示多种状态的错误

这样根据错误级别来判断是否显示错误的方法有利于面向不同的用户显示不同的信息。如果是面向开发人员，尽可能地显示出更多的调试信息；如果是普通用户，可以只显示普通的提示信息。

如图 16.18 所示为 Status、MultiStatus 和 ErrorDialog 对象之间的关系示意图，以便加深读者的理解。

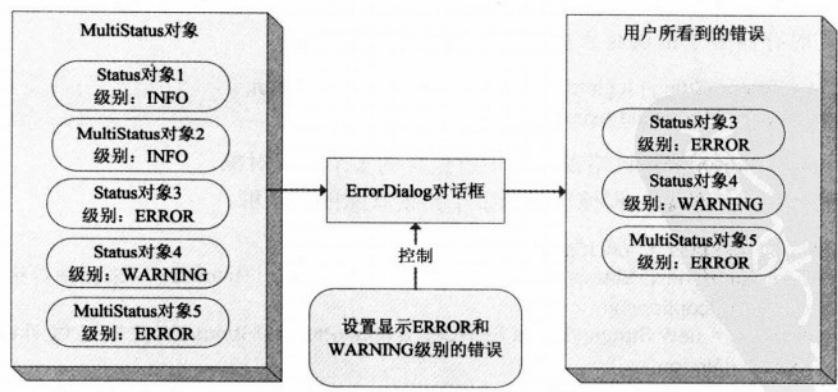


图 16.18 Status、MultiStatus 和 ErrorDialog 对象之间的关系示意图

从图 16.18 中可以看出 `ErrorDialog` 好比是一个过滤器, 可以通过一个开关过滤给用户显示不同的错误消息。

16.6 带有进度条的对话框 (`ProgressMonitorDialog`)

通常在进行费时的操作时, 要使用进度条告诉用户后台程序的运行情况, 在之前的学习中, 曾经在应用程序窗口的状态栏中显示进度状态。`JFace` 中也可以使用进度条对话框 (`ProgressMonitorDialog` 类) 来显示进度情况。如图 16.19 所示为 Eclipse 工作台在搜索文件时显示的带有进度条的对话框。

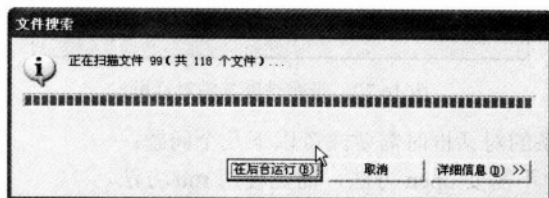


图 16.19 Eclipse 工作台带有进度条的对话框

下面就以一个示例程序来说明如何使用进度条的对话框。调用该对话框的办法与之前调用对话框的办法相同, 通过单击一个按钮来运行对话框, 在此只显示创建进度对话框的代码, 程序的其他部分请参阅本书所附的光盘部分。

`ProgressMonitorDialogTest.java`

```
try {
    ProgressMonitorDialog progressDialog = new ProgressMonitorDialog(Display.getCurrent().
getActive Shell());
    // 创建后台线程对象
    IRunnableWithProgress runnable = new IRunnableWithProgress() {
        // 线程运行的代码部分
        public void run(IPrimaryProgressMonitor monitor) throws InvocationTargetException, InterruptedException {
            // 设置显示在 UI 界面上的线程信息
            monitor.beginTask("开始执行任务...", 100);
            // 该线程在用户没有取消操作的情况下循环 10 次, 并且每次循环后设置进度增加 10, 表示一个任务已完成
            for (int i = 0; i < 10 && !monitor.isCanceled(); i++) {
                Thread.sleep(500); // 为了模拟耗时的操作, 每次循环后让该线程休息半秒钟
                monitor.worked(10); // 进度增加 10
                monitor.subTask("已完成第" + i + "个任务"); // 显示任务状态
            }
            // 循环完成后设置此任务已完成
            monitor.done();
            // 如果此时为用户取消的操作
            if (monitor.isCanceled())
```

```

        throw new InterruptedException("用户已取消了操作");
    }
};
progressDialog.run(true, true, runnable); // 启动线程
} catch (Exception ee) {
    ee.printStackTrace();
}
}

```

程序运行后的效果如图 16.20 所示。

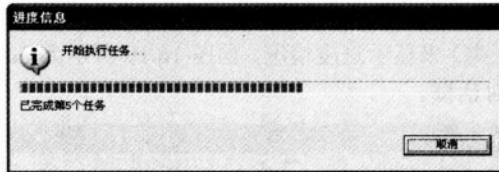


图 16.20 带有进度条的对话框

在使用带有进度条的对话框时需要注意以下几个问题：

- ❑ 打开对话框中不需要 open 方法，而是使用 run 方法。

```
run(boolean fork, boolean cancelable, IRunnableWithProgress runnable)
```

各参数的意义如下所示：

- ◆ fork：是否开辟另外一个线程执行。
- ◆ cancelable：是否可取消执行的线程。
- ◆ runnable：线程所执行的具体代码部分。
- ❑ 在后台运行程序的过程中，后台线程与前台界面的交互是通过 IProgressMonitor 对象来转化实现的，因为在 SWT 中是不允许其他线程访问 UI 线程的，而只能利用 UI 线程来调用。所以，使用 IProgressMonitor 对象可以不用考虑这些细节。
- ❑ 另外 ProgressDialog 除了 run() 方法外，还有一些常用的方法：
 - ◆ aboutToRun()：在运行线程之前所调用的方法。
 - ◆ finishedRun()：在运行的线程完成后调用的方法。
 - ◆ setCancelable(boolean cancelable)：设置是否显示“取消”按钮。
 - ◆ setOperationCancelButtonEnabled(boolean b)：设置“取消”按钮的状态，可用或不可用。
 - ◆ getProgressMonitor()：获得运行时的 IProgressMonitor 对象。

16.7 自定义对话框

以上介绍的对话框是最常用的对话框，但在实际项目的开发过程中，一般情况下这些对话框并不能满足需要，这时就需要自定义对话框。如图 16.21 所示为自定义的一个登录对话框。

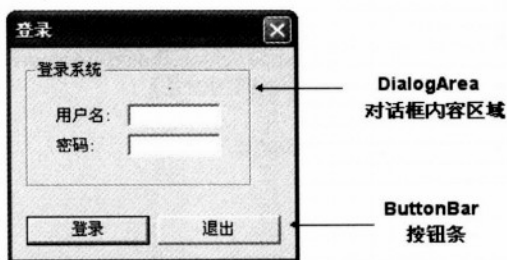


图 16.21 自定义的登录对话框

16.7.1 自定义对话框程序示例

下面就以如图 16.21 所示的对话框为例来具体讲述如何自定义对话框，首先看该对话框的代码：

LoginDialog.java

```
package com.fengmanfei.ch16;

import org.eclipse.jface.dialogs.Dialog;
import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.*;

public class LoginDialog extends Dialog {
    //定义按钮的常量
    public static final int LOGIN_ID = 0;
    public static final int LOGOUT_ID = 1;
    public static final String LOGIN_LABEL = "登录";
    public static final String LOGOUT_LABEL = "退出";
    //用户名和"密码"文本框
    private Text userName;
    private Text password;
    public LoginDialog(Shell parentShell) {
        super(parentShell);
    }

    protected void configureShell(Shell newShell) {
        super.configureShell(newShell);
        newShell.setText("系统登录");
    }
}

/*
 * (非 Javadoc)
 *
 * @see org.eclipse.jface.dialogs.Dialog#createDialogArea(org.eclipse.swt.widgets.Composite)
 * 覆盖该方法，在此方法中创建对话框的内容区域
 */
```

```

*/
protected Control createDialogArea(Composite parent) {
    Composite comp = (Composite) super.createDialogArea(parent);
    Group group = new Group(comp, SWT.NONE);
    group.setText("登录系统");
    GridLayout layout = new GridLayout();
    layout.marginHeight = 20;
    layout.marginWidth = 20;
    layout.numColumns = 2;
    group.setLayout(layout);
    new Label(group, SWT.NONE).setText("用户名: ");
    userName = new Text(group, SWT.BORDER | SWT.SINGLE);
    new Label(group, SWT.NONE).setText("密码: ");
    password = new Text(group, SWT.BORDER | SWT.SINGLE);
    password.setEchoChar('*');
    return parent;
}
/*
 * (非 Javadoc)
 *
 * @see org.eclipse.jface.dialogs.Dialog#createButtonsForButtonBar(org.eclipse.swt.
widgets.Composite)
 * 覆盖该方法，在此方法中创建所需要的按钮
 */
protected void createButtonsForButtonBar(Composite parent) {
    //使用父类中创建按钮的方法创建"登录"和"退出"按钮
    createButton(parent, LoginDialog.LOGIN_ID, LoginDialog.LOGIN_LABEL, true);
    createButton(parent, LoginDialog.LOGOUT_ID, LoginDialog.LOGOUT_LABEL, false);
}
/*
 * (非 Javadoc)
 *
 * @see org.eclipse.jface.dialogs.Dialog#buttonPressed(int)
 * 当单击对话框中的按钮时，调用此方法
 */
protected void buttonPressed(int buttonId) {
    //如果此时单击了"登录"按钮
    if (LoginDialog.LOGIN_ID == buttonId)
        System.out.println("登录！用户名为" + userName.getText() + "，密码为" + password.
getText());
    //如果此时单击了"取消"按钮
    else if (LoginDialog.LOGOUT_ID == buttonId)
        close();
}
}

```


16.7.2 自定义对话框的步骤

在创建自定义对话框的过程中，需要注意以下方面：

- ❑ 要继承自 `Dialog` 对象，一个对话框由对话框区域和按钮条组成。对话框区域是放置对话框各控件的区域，按钮条是放置该对话框按钮的区域。实现自定义对话框最基本的步骤是要覆盖以下几个方法：
 - ◆ `Control createDialogArea(Composite parent)`：创建对话框的各种控件。
 - ◆ `void createButtonsForButtonBar(Composite parent)`：创建对话框的按钮。
 - ◆ `void buttonPressed(int buttonId)`：处理按钮单击事件。
- ❑ 在覆盖 `createButtonsForButtonBar` 方法时，创建的按钮是通过调用父类以下所示的方法来创建按钮的。

```
Button createButton(Composite parent,int id, String label,boolean defaultButton);
```

其中，`id` 表示按钮所制定的索引值 `label` 为按钮所显示的文本，`defaultButton` 表示是否设置为默认选中状态。通常将 `id` 和 `label` 设置为常量，在 JFace 中已经提供了一些常用的按钮常量。在 `org.eclipse.jface.dialogs.IDialogConstants` 类中，例如，要创建一个“确定”按钮的代码如下：

```
createButton(parent, IDialogConstants.OK_ID, IDialogConstants.OK_LABEL, true);
```

当然 `IDialogConstants` 类中还有其他的按钮常量，请读者查询该类的 API 文档。

- ❑ 要处理单击不同按钮后的事件，此时要在 `buttonPressed(int buttonId)` 中处理，其中 `buttonId` 为按钮的索引值。

为了调用该对话框，还需要创建一个打开该对话框按钮的窗口，这里只显示了打开窗口的代码，程序的其他部分请参阅光盘附带的源代码部分。

LoginDialogTest.java

```
protected Control createContents(Composite parent) {  
    Composite composite = new Composite( parent , SWT.NONE);  
    composite.setLayout( new GridLayout());  
    Button button = new Button( composite ,SWT.NONE);  
    button.setText("打开自定义对话框示例");  
    button.addSelectionListener( new SelectionAdapter(){  
        public void widgetSelected(SelectionEvent e) {  
            LoginDialog dialog = new LoginDialog( Display.getCurrent(). GetActive Shell() );  
            dialog.open();  
        }  
    });  
    return parent;  
}
```

16.8 本章小结

通过本章的学习，读者已经对 JFace 框架提供的各种对话框有了一定的了解。下面就来回顾一下本章所学习的各种对话框。

- ❑ 信息提示对话框（`MessageDialog`）和输入对话框（`InputDialog`）是最常用的对话框，其中输入对话框对输入验证判断有效性的设计模式很值得学习研究，可以在设计其他有关验证的模式时受到启发，希望读者仔细研究源代码。
- ❑ 带有提示信息的对话框（`TitleAreaDialog`）可以根据不同输入信息，提示用户该如何进行操作。在 Eclipse 平台中，这种对话框使用得非常多。
- ❑ 错误提示对话框（`ErrorDialog`）可以根据错误级别来过滤显示给用户的错误信息，但这种对话框与 Eclipse 平台的开发耦合度比较高。
- ❑ 带有进度条的对话框（`ProgressMonitorDialog`）可以显示后台线程的进度，也是非常重要的对话框，凡是涉及线程的程序设计都要倍加小心，因为线程产生的 bug 是很难发现的。

最后，如果以上的这些对话框仍不能满足用户需求，此时可以自定义对话框，创建适合的对话框。



第 17 章 向导式对话框

本章将详细讲述如何使用 JFace 框架的向导式对话框。向导式对话框可以将较多的操作分步进行，并根据用户的选择指向不同的显示页面。使用向导式对话框可使程序的设计更加人性化，用户界面更加友好。

17.1 向导式对话框概述

在 Eclipse 的工作平台中选择“文件”|“新建”|“项目”命令，可以打开“新建项目”对话框，如图 17.1 所示，这是一个向导式对话框。

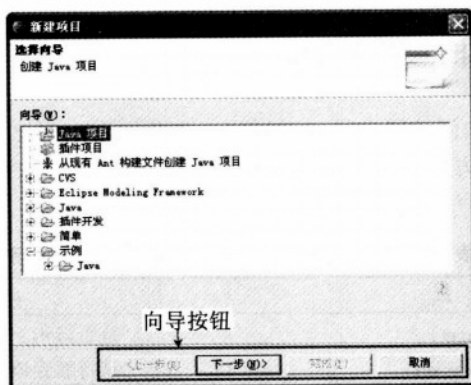


图 17.1 “新建项目”对话框

向导式对话框的主体部分类似于第 16 章学习的带提示的对话框（TitleAreaDialog），但与之不同的是，向导式对话框提供了一组向导按钮。当用户完成该页面进行设置时，可以单击“下一步”按钮进行下一步的设置，也可以单击“完成”按钮，则剩下页面的设置按照默认的设置完成。若此时单击“上一步”按钮，返回之前所设置的页面，重新设置。另外，也可以单击“取消”按钮，取消所有的操作。当然如果需要可以添加“帮助”按钮。

17.1.1 向导式对话框所涉及的类

向导式对话框所涉及的类都在 `org.eclipse.jface.wizard` 包中。为了更好地理解如何创建向导式对话框，首先认识以下 3 个接口：

- **IWizardPage**：表示一个向导页，一个向导过程是由多个向导页面组成的。

- ❑ IWizard: 负责管理该向导所有的向导页 (IWizardPage)。
- ❑ IWizardContainer: 显示向导内容 (IWizard) 的容器。

在 JFace 中, 实现这 3 个接口的类对应的分别是 WizardPage、Wizard 和 WizardDialog 类。如图 17.2 所示为这 3 个类与其实现接口的关系示意图。

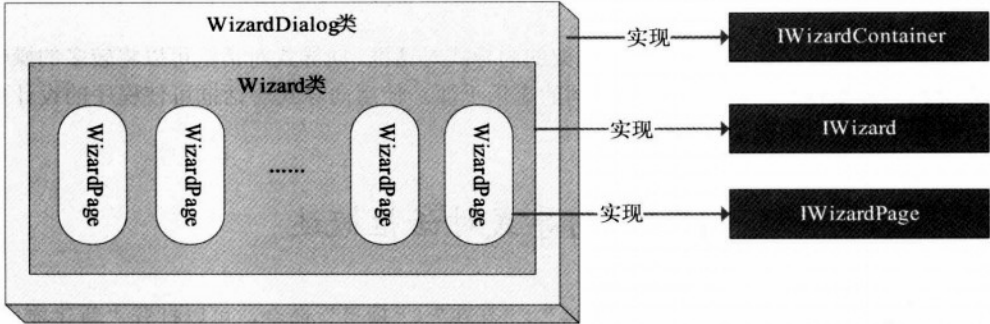


图 17.2 WizardPage、Wizard 和 WizardDialog 类关系示意图

17.1.2 向导式对话框的常用方法

为了更清楚地理解如何使用向导式对话框, 先分别看一下这 3 个类中的主要方法。WizardPage 类中的主要方法如表 17.1 所示, Wizard 类中的主要方法如表 17.2 所示, WizardDialog 类中的主要方法如表 17.3 所示。

表 17.1 WizardPage 类中的主要方法及其说明

方法名称	说明
boolean canFlipToNextPage()	是否显示下一页按钮
IWizardContainer getContainer()	获得放置该页的容器
IDialogSettings getDialogSettings()	获得 IDialogSettings 对象
Image getImage()	获得该页的图标对象
String getName()	获得该页的名称标识
IWizardPage getNextPage()	获得下一页
IWizardPage getPreviousPage()	获得上一页
IWizard getWizard()	获得此页所在 IWizard 对象
boolean isCurrentPage()	此页是否为当前显示的页
boolean isPageComplete()	该页面是否已完成状态
void setDescription(String description)	设置该页面的描述信息
void setPageComplete(boolean complete)	设置该页面的状态
void setPreviousPage(IWizardPage page)	设置上一页
void setWizard(IWizard newWizard)	设置该页所属的 IWizard 对象

表 17.2 Wizard 类中的主要方法及其说明

方 法 名 称	说 明
void addPage(IWizardPage page)	向该向导添加页面
void addPages()	添加多个页面，通常由子类实现
boolean canFinish()	是否激活“完成”按钮
void createPageControls(Composite pageContainer)	创建各个向导页面的控件
IWizardContainer getContainer()	获得该向导所在的容器
Image getDefaultPageImage()	获得默认页的图标
IDialogSettings getDialogSettings()	获得该向导所对应的 IDialogSettings 对象
IWizardPage getNextPage(IWizardPage page)	获得某一指定页面的下一页
IWizardPage getPreviousPage(IWizardPage page)	获得某一指定页面上的一页
IWizardPage getPage(String name)	获得某一指定页面名称的页面
int getPageCount()	获得总共的页面数
IWizardPage[] getPages()	获得所有的页面对象
IWizardPage getStartingPage()	获得该向导的首页
RGB getTitleBarColor()	获得标题区域的颜色
String getWindowTitle()	获得窗口的标题名
boolean isHelpAvailable()	是否显示帮助功能
boolean needsPreviousAndNextButtons()	是否需要显示上一步和下一步操作
boolean needsProgressMonitor()	是否需要显示进度条
boolean performCancel()	是否响应取消操作
abstract boolean performFinish()	响应完成操作，该方法需由子类来实现
void setContainer(IWizardContainer wizardContainer)	设置该向导所在的容器对象
void setDefaultPageImageDescriptor(ImageDescriptor imageDescriptor)	设置默认页面的图标
void setDialogSettings(IDialogSettings settings)	设置该向导的 IDialogSettings 对象
void setForcePreviousAndNextButtons(boolean b)	设置强制显示上一步和下一步操作，即使只有一个页面时
void setHelpAvailable(boolean b)	设置是否启用帮助操作
void setNeedsProgressMonitor(boolean b)	设置是否需要进度条

表 17.3 WizardDialog 类中的主要方法及其说明

方 法 名 称	说 明
IWizardPage getCurrentPage()	获得当前的页面
Shell getShell()	获得显示该向导的窗口
void showPage(IWizardPage page)	显示指定的页面
void updateButtons()	改变向导按钮的状态
void updateMessage()	更改显示当前页面的提示信息
void updateTitleBar()	改变标题区域的内容
void updateWindowTitle()	更改窗口标题
updateSize()	改变当前向导对话框的大小

17.2 简单的向导式对话框示例

下面以一个读者调查问卷表的程序来实现简单的向导对话框。该调查问卷一共有两道题目，每道题为一个向导页面，另外还有一个感谢页面。当用户完成一道题目后，单击“下一步”按钮，进入下一道问题的页面，当答完全部页面后，出现感谢对话框，此时可以单击“完成”按钮。

17.2.1 第一个问题向导页面

首先将第一个问题作为第一个向导页面，如图 17.3 所示为第一个问题页面运行后的效果图。

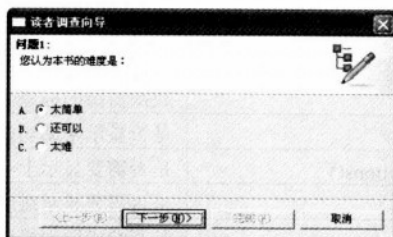


图 17.3 第一个问题向导页面运行效果图

创建这样一个向导页面的代码如下：

QuestionOne.java

```
package com.fengmanfei.ch17;

import org.eclipse.jface.resource.ImageDescriptor;
import org.eclipse.jface.wizard.WizardPage;
import org.eclipse.jface.dialogs.*;
import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.*;

public class QuestionOne extends WizardPage{
    public QuestionOne() {
        super(BookSurveyWizard.Q1,"问题 1:",ImageDescriptor.createFromFile (Question One.
class,"q.gif"));
        //设置标题消息
        this.setMessage("您认为本书的难度是：");
    }
    //该方法为必须实现的方法，在该方法中创建向导页面的控件
    public void createControl(Composite parent) {
        Composite composite = new Composite(parent, SWT.NONE);
```

```

composite.setLayout(new GridLayout(2,false));
new Label(composite, SWT.LEFT).setText("A.");
Button b1 = new Button( composite, SWT.RADIO);
b1.setText("太简单");
b1.setSelection(true);
new Label(composite, SWT.LEFT).setText("B.");
Button b2 = new Button( composite, SWT.RADIO);
b2.setText("还可以");
new Label(composite, SWT.LEFT).setText("C.");
Button b3 = new Button( composite, SWT.RADIO);
b3.setText("太难");
//该方法非常重要，否则不能显示设置的控件
setControl(composite);
}
}

```

17.2.2 第二个问题向导页面

第二个问题页面创建的方式与第一个页面类似，如图 17.4 所示为第二个页面运行后的效果示意图。

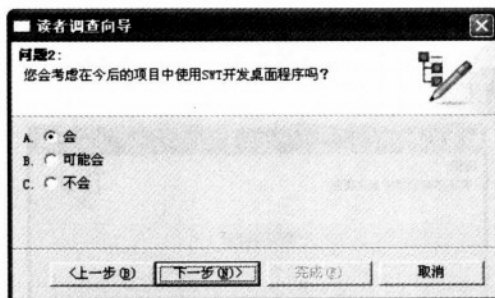


图 17.4 第二个问题向导页面运行效果图

创建第二个问题向导页面的代码如下：

QuestionTwo.java

```

package com.fengmanfei.ch17;

import org.eclipse.jface.resource.ImageDescriptor;
import org.eclipse.jface.wizard.WizardPage;
import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.*;

public class QuestionTwo extends WizardPage{
    public QuestionTwo() {
        super(BookSurveyWizard.Q2,"问题 2:",ImageDescriptor.createFromFile(Question One.

```

```

class,"q.gif"));
    this.setMessage("您会考虑在今后的项目中使用 SWT 开发桌面程序吗? ");
}
public void createControl(Composite parent) {
    Composite composite = new Composite(parent, SWT.NONE);
    composite.setLayout(new GridLayout(2, false));
    new Label(composite, SWT.LEFT).setText("A.");
    Button b1 = new Button( composite, SWT.RADIO);
    b1.setText("会");
    b1.setSelection(true);
    new Label(composite, SWT.LEFT).setText("B.");
    Button b2 = new Button( composite, SWT.RADIO);
    b2.setText("可能会");
    new Label(composite, SWT.LEFT).setText("C.");
    Button b3 = new Button( composite, SWT.RADIO);
    b3.setText("不会");
    setControl(composite);
}
}

```

17.2.3 感谢向导页面

最后在调查结束后，表示对用户参加调查的感谢。此时，“完成”按钮处于可用状态，用户单击“完成”按钮结束本次调查。感谢页面运行后的效果如图 17.5 所示。

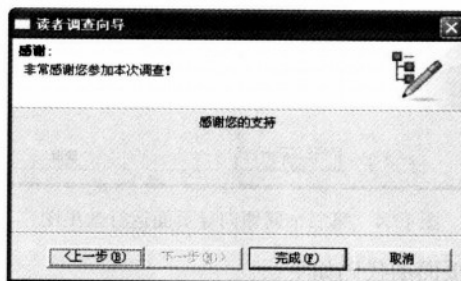


图 17.5 感谢向导页面运行效果图

该感谢向导页面实现的代码如下：

Thanks.java

```

package com.fengmanfei.ch17;

import org.eclipse.jface.resource.ImageDescriptor;
import org.eclipse.jface.wizard.WizardPage;
import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.FillLayout;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Label;

```

```
public class Thanks extends WizardPage {

    protected Thanks() {
        super(BookSurveyWizard.THANKS, "感谢:", ImageDescriptor.createFromFile(QuestionOne.class, "q.gif"));
        this.setMessage("非常感谢您参加本次调查!");
    }

    public void createControl(Composite parent) {
        Composite composite = new Composite(parent, SWT.NONE);
        composite.setLayout(new FillLayout());
        new Label(composite, SWT.CENTER).setText("感谢您的支持");
        setControl(composite);
    }
}
```

通过以上 3 个向导页面，可以总结出创建一个向导页面所需要注意的问题：

- ❑ 每一个向导页面都要继承自 WizardPage 类，并需要实现 createControl 抽象方法。
- ❑ 在构造向导页面时要使用 WizardPage 类构造方法。WizardPage 类的构造方法有两种：
 - ◆ WizardPage(String pageName): 其中 pageName 为该页名称，一般使用字符串常量。
 - ◆ WizardPage(String pageName, String title, ImageDescriptor titleImage): 其中，pageName 为该页名称，title 为显示的标题，titleImage 为右侧显示的图标文件。
- ❑ 在 createControl 方法中创建向导对话框的控件后，不要忘记调用 setControl 方法，否则将不能显示控件。
- ❑ 另外，如果想为该页面进行更多的设置，可以覆盖父类中的一些方法。比如若想为该页面添加帮助，需覆盖 performHelp 方法。若需要判断是否将“下一步”状态置为可用，需要覆盖 canFlipToNextPage 方法。这些都涉及复杂的向导控制，有关这些用法将在 17.4 节进行讲解。

17.2.4 创建向导

有了每个向导页面，下面就需要将这些页面组装起来，以下为将各个向导页面组装起来的具体实现代码：

BookSurveyWizard.java

```
package com.fengmanfei.ch17;
import org.eclipse.jface.wizard.Wizard;
public class BookSurveyWizard extends Wizard {

    public static final String Q1 = "QUESTION_1";
    public static final String Q2 = "QUESTION_2";
    public static final String THANKS = "THANKS";
    //声明向导的 3 个页面
```

```
private QuestionOne one;
private QuestionTwo two;
private Thanks thanks;
public BookSurveyWizard(){
    //创建 3 个向导页面对象
    one = new QuestionOne();
    two = new QuestionTwo();
    thanks = new Thanks();
    //分别添加这 3 个页面
    this.addPage( one );
    this.addPage( two );
    this.addPage( thanks );
    this.setWindowTitle("读者调查向导");//向导标题
}
/* (非 Javadoc)
 * @see org.eclipse.jface.wizard.Wizard#canFinish()
 * 确定“完成”按钮是否可用, true 为可用, false 为不可用
 */
public boolean canFinish() {
    //仅当当前页面为感谢页面时才将“完成”按钮置为可用状态
    if (this.getContainer().getCurrentPage() == thanks )
        return true;
    else
        return false;
}
//必须实现该方法, 当单击“完成”按钮后调用此方法
public boolean performFinish() {
    return true;
}
}
```

从以上程序的代码中可以看出, 创建组装各个向导页面类时应该注意以下几个问题:

- ☐ 该向导必须继承自 Wizard 类, 必须实现 performFinish 方法, 该方法为单击“完成”按钮后调用的方法。
- ☐ 将各个向导页面添加到向导中使用的是 addPage 方法。注意, 一定要按照向导页面的顺序添加到向导中。
- ☐ 可以通过覆盖父类中的 canFinish 方法, 设置“完成”按钮的可用状态。
- ☐ 另外, 父类中仍有一些方法, 可以在需要时通过覆盖来实现。比如, 要判断“取消”按钮的状态可以覆盖 performCancel 方法等。
- ☐ 另外还有一些 Wizard 类常用的方法, 如设置“帮助”按钮是否可用 setHelp, Available 设置是否显示进度条 setNeedsProgressMonitor 等。

17.2.5 创建测试程序

最后要编写一个测试窗口来调用该向导对话框。该测试程序的代码如下:

WizardTest.java

```

package com.fengmanfei.ch17;
import org.eclipse.jface.window.ApplicationWindow;
import org.eclipse.jface.wizard.WizardDialog;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.SelectionAdapter;
import org.eclipse.swt.events.SelectionEvent;
import org.eclipse.swt.layout.RowLayout;
import org.eclipse.swt.widgets.*;

public class WizardTest extends ApplicationWindow{

    public WizardTest() {
        super(null);
    }
    protected Control createContents(Composite parent) {
        parent.setLayout( new RowLayout(SWT.VERTICAL));
        Button button = new Button( parent ,SWT.NONE);
        button.setText("打开简单向导对话框");
        button.addSelectionListener( new SelectionAdapter(){
            public void widgetSelected(SelectionEvent e) {
                //调用该对话框
                WizardDialog dlg = new WizardDialog(Display.getCurrent().getActiveShell(),
new BookSurveyWizard());
                dlg.open();
            }
        });
        return parent;
    }
    public static void main(String[] args) {
        WizardTest test = new WizardTest();
        test.setBlockOnOpen( true );
        test.open();
        Display.getCurrent().dispose();
    }
}

```

在调用向导式对话框时，使用的构造方法是：

```
WizardDialog(Shell parentShell, IWizard newWizard)
```

其中，parentShell 为对话框的父窗口，newWizard 为该窗口的向导对象。创建完向导对话框后，调用 open 方法打开窗口。创建向导式对话框的简单代码如下所示：

```

WizardDialog dlg = new WizardDialog(Display.getCurrent().getActiveShell(), new BookSurvey
Wizard());
dlg.open();

```

17.3 保存对话框状态

在使用向导式的对话框时，有时可能需要设置向导页面的默认状态。例如，在 Eclipse 菜单中选择“新建”|“类”命令，此时创建类时选择“是否要创建 main 方法”选项后，下次再创建新类时，该选项自动被选中，也就是说，每次创建新类时都使用上一次创建类时的默认设置。

当一个对话框关闭时，这个对象就销毁了，若要想保存当前的设置，这就涉及数据持久化的问题。JFace 的对话框中提供了 `org.eclipse.jface.dialogs.DialogSettings` 类，可以轻松地将设置保存到文件中，实现数据的持久化。

使用 `DialogSettings` 可以将设置保存为 xml 的文件格式，每个对应的设置为一一对应的 key 和 value 的形式。例如，创建以下代码所示的程序：

DialogSettingTest.java

```
package com.fengmanfei.ch17;
import java.io.IOException;
import org.eclipse.jface.dialogs.DialogSettings;
public class DialogSettingTest {

    public static void main(String[] args) {
        //创建对话框设置对象
        DialogSettings settings = new DialogSettings("survey");
        //放入 key 和 value
        settings.put("Q1",0);
        settings.put("Q2",1);
        settings.put("Q3",2);
        //保存到文件中
        try {
            settings.save("f:\\dialog.xml");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

该程序运行后保存的 xml 文件的内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<section name="survey">
  <item key="Q2" value="1"/>
  <item key="Q1" value="0"/>
  <item key="Q3" value="2"/>
</section>
```

当保存一个设置的值时，只需要使用 `put` 方法设置，然后调用 `save` 方法进行保存。当

然这只是保存设置的过程。若要读取 xml 文件中设置的信息，只需要使用 `get` 方法即可。例如，以下代码为读取默认值的方法。

```
DialogSettings settings = new DialogSettings( null );
try {
    settings.load( "f:\\dialog.xml" );
} catch (IOException e) {
    e.printStackTrace();
}
System.out.println("Q1="+settings.get("Q1"));
System.out.println("Q2="+settings.get("Q2"));
System.out.println("Q3="+settings.get("Q3"));
```

在使用向导式对话框时可以直接设置向导所使用的 `DialogSettings` 对象，使用 `setDialogSettings` 方法。获得向导的 `DialogSettings` 对象的方法是 `getDialogSettings` 方法。

```
BookSurveyWizard wiz = new BookSurveyWizard();
DialogSettings settings = new DialogSettings(null);
try {
    settings.load("f:\\dialog.xml");
} catch (IOException ee) {
    ee.printStackTrace();
}
wiz.setDialogSettings( settings );
```

17.4 复杂的向导式对话框示例

在实际的项目开发中，很少有像流水账式的向导页面。更多的情况是，当前的页面设置会决定下一个所要显示的页面。下面就改进一下之前编写的调查问卷的程序。

为了能够对读者进行联系，这里设计调查增加一个页面，让读者进行选择，如果读者愿意留下联系方式，则可以留下邮件后，完成调查；否则，直接完成调查。图 17.6 所示为该调查问卷系统向导的流程图。

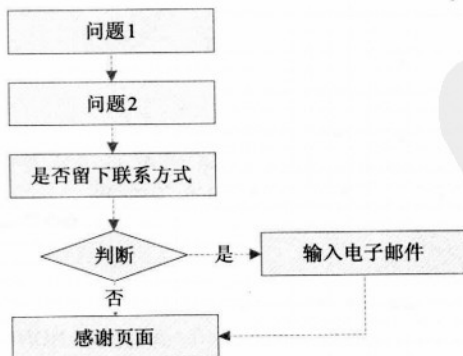


图 17.6 改进功能后的向导页面流程图

17.4.1 自定义向导页面

根据图 17.6 所示的页面流程的分析, 要增加两个向导页面, 一个页面是询问读者是否愿意留下联系方式的页面, 一个页面是输入电子邮件的页面。

1. 询问读者是否留下联系方式的页面

运行后效果如图 17.7 所示。当选择“否”选项时, 单击“下一步”按钮后, 将进入到感谢页面。而当选择“是”时, 将进入到输入 Email 的对话框页面。

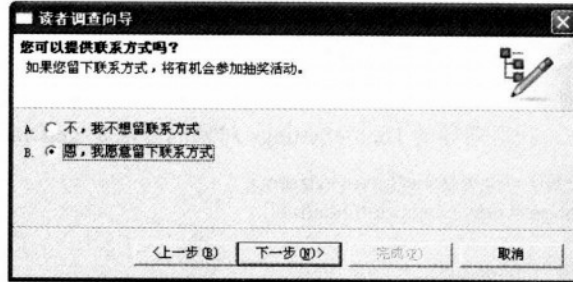


图 17.7 询问是否愿意留下联系方式向导页面

实现该向导页面的代码如下:

AskForContact.java

```
package com.fengmanfei.ch17;

import org.eclipse.jface.resource.ImageDescriptor;
import org.eclipse.jface.wizard.IWizardPage;
import org.eclipse.jface.wizard.WizardPage;
import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Label;

public class AskForContact extends WizardPage {
    private Button no;
    private Button yes;
    protected AskForContact() {
        super(BookSurveyWizard.ASK_FOR_CONTACT, "您可以提供联系方式吗?",
            ImageDescriptor.createFromFile(QuestionOne.class, "q.gif"));
        this.setMessage("如果您留下联系方式, 将有机会参加抽奖活动。");
    }

    public void createControl(Composite parent) {
        Composite composite = new Composite(parent, SWT.NONE);
        composite.setLayout(new GridLayout(2, false));
        new Label(composite, SWT.LEFT).setText("A.");
```

```

        no = new Button(composite, SWT.RADIO);
        no.setText("不, 我不想留联系方式");
        no.setSelection(true);
        new Label(composite, SWT.LEFT).setText("B.");
        yes = new Button(composite, SWT.RADIO);
        yes.setText("恩, 我愿意留下联系方式");
        setControl(composite);
    }
    //判断是否需要输入联系方式
    public boolean canContact() {
        if (yes.getSelection() == true)
            return true;
        return false;
    }
    /*
     * (非 Javadoc)
     *
     * @see org.eclipse.jface.wizard.WizardPage#getNextPage()
     * 覆盖父类中的该方法可以设置下一页所显示的页面
     */
    public IWizardPage getNextPage() {
        //如果留下联系方式, 则返回下一页
        if (canContact())
            return super.getNextPage();
        //否则返回感谢页面
        return getWizard().getPage(BookSurveyWizard.THANKS);
    }
}

```

实现能够自定义下一步的页面的方法主要是覆盖父类中的 `getNextPage` 方法。通过 `getWizard()` 方法可以获得该页面所在的向导对象, 然后再通过对对象的 `getPage` 方法获得指定下一步所显示的页面。

2. 输入电子邮件的页面

运行后效果如图 17.8 所示, 进入该页面时, “下一步” 按钮是不可用状态, 当输入正确的 Email 后, “下一步” 按钮才置为可用状态。

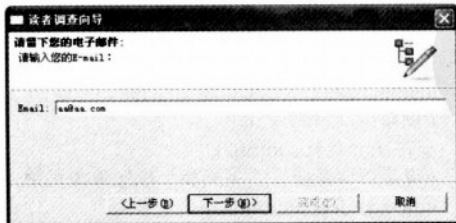


图 17.8 输入电子邮件的向导页面的效果图

实现该向导页面的代码如下：

Email.java

```
package com.fengmanfei.ch17;

import org.eclipse.jface.resource.ImageDescriptor;
import org.eclipse.jface.wizard.WizardPage;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.VerifyEvent;
import org.eclipse.swt.events.VerifyListener;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.Text;

public class Email extends WizardPage {

    private Text email;

    protected Email() {
        super(BookSurveyWizard.EMAIL, "请留下您的电子邮件：", ImageDescriptor.create
        FromFile(QuestionOne.class, "q.gif"));
        this.setMessage("请输入您的 E-mail: ");
        //初始状态时设置页面未完成，“下一步”按钮置为不可用
        this.setPageComplete(false);
    }

    public void createControl(Composite parent) {
        Composite composite = new Composite(parent, SWT.NONE);
        composite.setLayout(new GridLayout(2, false));
        new Label(composite, SWT.LEFT).setText("Email:");
        email = new Text(composite, SWT.BORDER);
        email.setLayoutData(new GridData(GridData.FILL_HORIZONTAL));
        //为电子邮件文本框注册事件
        email.addVerifyListener(new VerifyListener() {
            //当文本框字符改变时
            public void verifyText(VerifyEvent e) {
                //如果此时符合 Email 的格式
                if (email.getText().indexOf('@') > -1) {
                    //设置错误消息为 null
                    setErrorMessage(null);
                    //设置页面完成，“下一步”按钮置为可用
                    setPageComplete(true);
                } else
                    //如果不符合 Email 格式，则提示错误消息
                    setErrorMessage("请输入有效的 Email!");
            }
        });
    }
}
```

```

        setControl(composite);
    }
}

```

注意, 在该程序中, 可以通过 `setPageComplete` 方法来设置“下一步”按钮的是否可用状态。最后, 只需将这两个向导页面添加到向导对象中, 代码如下:

```

public BookSurveyWizard(){
    //.....程序省略
    this.addPage( new AskForContact());
    this.addPage( new Email());
}

```

然后再运行此向导对话框, 就可以看到自定义的向导页面的调查表了。

17.4.2 为向导添加帮助

另外, 还可以为每个向导页面添加“帮助”按钮。如图 17.9 所示为第一个问题向导页面添加了“帮助”按钮后, 弹出的一个帮助对话框。

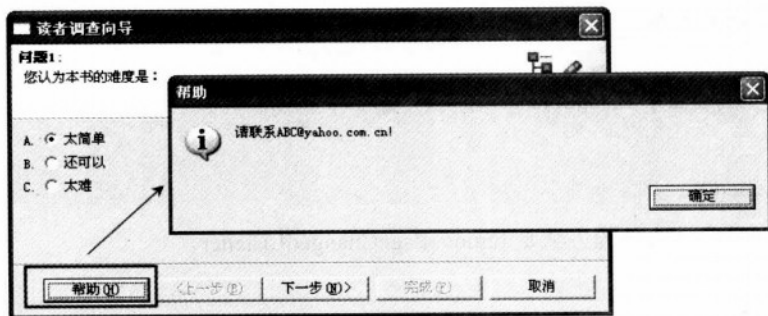


图 17.9 添加“帮助”按钮后的向导对话框

添加“帮助”按钮的方法如下:

(1) 首先在 `BookSurveyWizard` 类的构造方法中设置“帮助”按钮可用:

```

public BookSurveyWizard(){
    //.....程序省略
    this.setHelpAvailable( true );
}

```

(2) 然后分别设置每个页面所指向的帮助操作。例如, 在第一个问题页面, 要弹出一个对话框显示帮助信息, 需要在 `QuestionOne` 这个类中, 覆盖父类的 `performHelp` 方法。代码如下:

```

public class QuestionOne extends WizardPage{
    //.....代码省略
    /* (非 Javadoc)
     * @see org.eclipse.jface.dialogs.DialogPage#performHelp()
     */
}

```

```
*/
public void performHelp() {
    MessageDialog.openInformation(Display.getCurrent().getActiveShell(), "帮助", "请联系
ABC@yahoo.com.cn!");
}
}
```

要为一个向导页面添加“帮助”按钮所触发的操作，需要覆盖父类中的 `performHelp` 方法。

17.5 向导式对话框的事件处理

对于向导式对话框，可以为该对象注册页面改变事件（`PageChangedEvent`），也就是当转到上一页或下一页时所触发的事件。如果为该向导对话框设置了 `DialogSettings` 对象，需要在页面转换时，保存当前页面的设置，那在该事件触发时保存比较合适。例如，以下代码为向导式对话框注册了页面改变事件：

```
WizardDialog dlg = new WizardDialog(Display.getCurrent().getActiveShell(), new BookSurvey
Wizard());
dlg.addPageChangeListener( new IPageChangeListener(){
    public void pageChanged(PageChangedEvent event) {
        IWizardPage page = (IWizardPage) event.getSelectedPage();
        //可以保存 DialogSettings 的一些设置
    }
});
dlg.open();
```

另外，要移除该事件的方法是 `removePageChangeListener`。

17.6 本章小结

通过本章的学习，读者对更高级的对话框——向导式对话框有了一定的了解，现在回顾一下本章的内容。首先介绍了向导式对话框的整个构架，理解了 `WizardPage`、`Wizard` 和 `WizardDialog` 类之间的关系。然后，以一个流水账式的向导作为例子，讲述了如何创建一个简单的向导式对话框。接着讨论了保存对话框状态的 `DialogSettings` 类的使用。最后讲述了如何创建复杂向导，这包括如何自定义向导流程、为向导添加帮助和处理向导页面改变事件等。

第 18 章 首 选 项

本章将详细讲述 JFace 框架中的首选项（Preference）的设置。在一个应用系统中，首选项是可以根据用户的喜好来设定的，一般是以键（key）/值（value）来保存的。JFace 框架中已经提供了完善的设置首选项的功能。

18.1 首选项概述

在 Eclipse 工作平台中选择“窗口”|“首选项”命令，弹出 Eclipse 的“首选项”对话框，如图 18.1 所示。

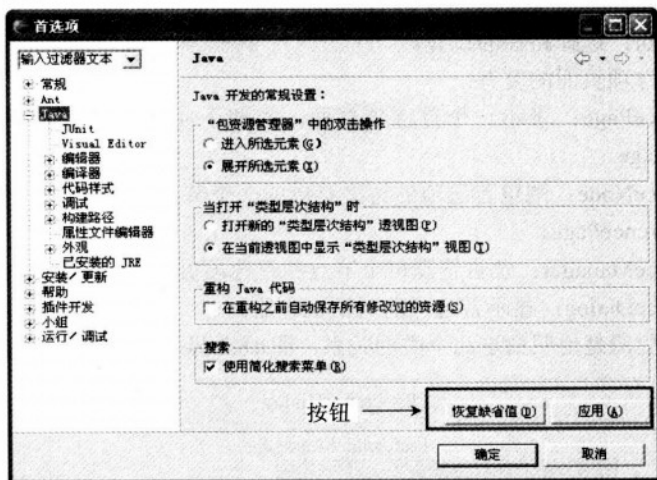


图 18.1 Eclipse 的“首选项”对话框

从图中可以看出，由于一个系统中需要设置的首选项比较多时，就需要将选项分成几个页面来显示，一个页面可以进行某一个主题的设置。图中右侧的区域即为首选项中的一个页面。在这个页面上，有“恢复缺省值”和“应用”两个按钮，单击“恢复缺省值”按钮时，恢复到该页面系统默认的设置值，而单击“应用”按钮时，将保存当前的设置。

为了便于浏览不同的首选项页面，左侧提供了树型的菜单，来进行浏览某一个首选项页面。当单击树的节点时，便显示该节点所对应的首选项页面。下面对一些常用的类作一下简单的介绍。有关保存首选项设置的类有：

- ❑ **PreferenceStore**：可以将设置以 key/value 的形式保存为文件，也可以载入文件后方便地获取所设定的值。

- ❑ **PreferenceConverter**: 保存为文件的首选项为字符串的形式, 对于像颜色、字体这些对象, 无法直接保存为字符串, 这时就需要 **PreferenceConverter** 类转换, 将颜色保存为字符串的格式, 并且将字符串转化为颜色对象。

有关首选项的设置类有:

- ❑ **FieldEditor**: 这是一个抽象类, 虽然对选项的设置可以使用最基本的标签、按钮等 SWT 控件实现, 但会很麻烦, 为了简化创建各种设置的选项, 可以使用 **FieldEditor** 对象创建常用的控件。
- ❑ **BooleanFieldEditor**: 布尔型选项的设置。
- ❑ **IntegerFieldEditor**: 整型值设置。
- ❑ **StringFieldEditor**: 字符串型值的设置。
- ❑ **RadioGroupFieldEditor**: 分组面板型的设置。
- ❑ **ColorFieldEditor**: 颜色值的设置。
- ❑ **FontFieldEditor**: 字体型值的设置。
- ❑ **DirectoryFieldEditor**: 选择文件目录的设置。
- ❑ **FileFieldEditor**: 选择文件的设置。
- ❑ **PathEditor**: 选择路径的设置。

有关显示首选项页面的类有:

- ❑ **PreferencePage**: 表示一个首选项页面, 类似于向导式对话框中的一个向导页面 **WizardPage**。
- ❑ **PreferenceNode**: 浏览首选项的树型菜单的一个节点, 一个节点对应一个首选项页面 **PreferencePage**。
- ❑ **PreferenceManager**: 负责管理每个节点和首选项页面, 包括树的结构等。
- ❑ **PreferenceDialog**: 显示首选项的对话框容器。

为了使读者更清楚地明白这几个类的关系, 图 18.2 显示了这几个类的关系示意图。

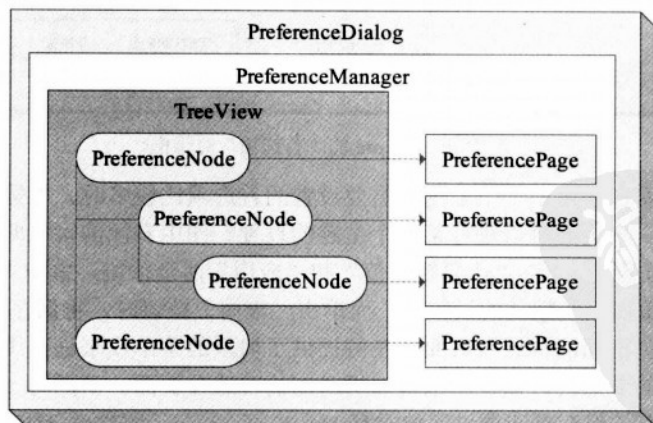


图 18.2 PreferencePage、PreferenceNode、PreferenceManager 和 PreferenceDialog 关系示意图

18.2 保存首选项的设置

在学习如何显示首选项页面之前,首先来认识一下保存首选项的类 `PreferenceStore`。Java 中使用 `.properties` 为扩展名的文件来保存首选项的设置。

18.2.1 首选项值的设置和获取

首选项的设置是以 `key` 和 `value` 格式来表示的。例如,以下所示了 3 个选项的设置:

```
Password=123
UserName=Janet
Database=mysql
```

要保存这样一个格式的文件代码如下:

```
PreferenceStore preferenceStore =
new PreferenceStore("F:\\myPreference.properties");//保存设置的文件名
preferenceStore.setValue("Database", "mysql");//设置 key 和 value 值
preferenceStore.setValue("UserName", "Janet");
preferenceStore.setValue("Password", "123");
try {
preferenceStore.save();//保存设置
} catch (IOException e) {
e.printStackTrace();
}
```

这是保存设置的过程,若要读取设置的值也非常简单,只需使用 `get` 方法获得。例如,要获得 `Database` 所设置的值代码如下:

```
PreferenceStore preferenceStore =
new PreferenceStore("F:\\myPreference.properties");
try {
preferenceStore.load();//装载文件
} catch (IOException e) {
e.printStackTrace();
}
String database = preferenceStore.getString("Database");
```

另外可根据获得的不同数据类型的值而使用不同的方法,如获得 `int` 型数据使用 `getInt(String name)` 方法,获得 `double` 型数据使用 `getDouble(String name)` 等。

最后, `PreferenceStore` 还提供了设置默认值的方法 `setDefaultXXX`, 可以为某一个选项设置默认值。当通过 `get` 方法获取一个设置时,程序首先查找保存的 `.properties` 文件中是否已设置了该值,如果没有则返回该选项的默认值。

18.2.2 保存首选项所涉及的事件

当某一个选项值改变时可以触发 `PropertyChangeEvent` 事件，通过 `addPropertyChangeListener` 方法可以为 `PreferenceStore` 对象注册事件。例如，以下是注册了选项改变事件的代码。

```
//注册改变选项事件
preferenceStore.addPropertyChangeListener(new IPropertyChangeListener()
public void propertyChange(PropertyChangeEvent event) {
//如果改变的是 Database 选项的值，则输出原来的值和新设定的值
if (event.getProperty().equals("Database")) {
System.out.println("old value:" + event.getOldValue());
System.out.println("new value:" + event.getNewValue());
}
}
});
referenceStore.setValue("Database", "sqlserver");//触发事件
```

18.3 显示首选项页面

理解了显示首选项这几个类的关系，下面就来学习如何将首选项页面显示出来。

18.3.1 创建一个首选项页面

与向导对话框类似，首选项对话框最基本的单元是一个首选项页面，如图 18.3 所示为本例所要创建的一个首选项页面。该页面由两个文本框和两个标签组成。

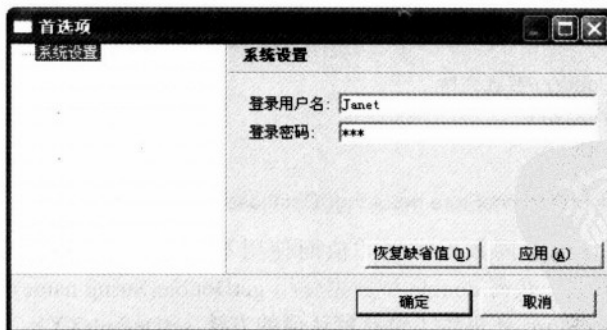


图 18.3 一个首选项页面效果图

创建这样一个首选项页面的代码如下：

SystemSettingPage.java

```
package com.fengmanfei.ch18;
import org.eclipse.jface.preference.IPreferenceStore;
import org.eclipse.jface.preference.PreferencePage;
import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.widgets.*;

public class SystemSettingPage extends PreferencePage {

    private Text userName;
    private Text password;
    //该方法为必须实现的方法，在此方法中创建页面上的各种控件
    protected Control createContents(Composite parent) {
        Composite composite = new Composite(parent, SWT.NONE);
        composite.setLayout(new GridLayout(2, false));
        //获取保存此页面的 PreferenceStore 对象
        IPreferenceStore preferenceStore = getPreferenceStore();

        new Label(composite, SWT.LEFT).setText("登录用户名:");
        userName = new Text(composite, SWT.BORDER);
        userName.setLayoutData(new GridData(GridData.FILL_HORIZONTAL));
        //设置用户名为保存在文件中的值
        userName.setText(preferenceStore.getString(Constants.USER_NAME));

        new Label(composite, SWT.LEFT).setText("登录密码:");
        password = new Text(composite, SWT.BORDER);
        password.setEchoChar('*');
        password.setLayoutData(new GridData(GridData.FILL_HORIZONTAL));
        //设置密码为保存在文件中的值
        password.setText(preferenceStore.getString(Constants.PASSWORD));
        return composite;
    }

    /*
     * (非 Javadoc)
     *
     * @see org.eclipse.jface.preference.PreferencePage#performDefaults()
     * 覆盖父类中的方法，当单击“恢复缺省值”按钮时调用该方法
     */
    protected void performDefaults() {
        IPreferenceStore preferenceStore = getPreferenceStore();
        userName.setText( preferenceStore.getDefaultString(Constants.USER_NAME));
        password.setText( preferenceStore.getDefaultString(Constants.PASSWORD));
    }

    /*
     * (非 Javadoc)
     */
}
```

```

*
* @see org.eclipse.jface.preference.PreferencePage#performOk()
* 覆盖父类中的方法，当单击“应用”按钮时调用该方法
*/
public boolean performOk() {
    IPreferenceStore preferenceStore = getPreferenceStore();
    if (userName != null)
        preferenceStore.setValue(Constants.USER_NAME, userName.getText());
    if (password != null)
        preferenceStore.setValue(Constants.PASSWORD, password.getText());
    return true;
}
}

```

从以上代码中可以总结出，创建一个首选项页面时需要注意以下问题：

- ❑ 每个首选项页面都要继承自 `PreferencePage` 类，而且必须实现 `createContents` 的抽象方法。这与之前学习使用的向导页面的方法类似。
- ❑ 在 `createContents` 方法中可以创建页面的各种控件。
- ❑ 覆盖父类中的 `performDefaults` 方法可以添加“恢复缺省值”按钮的操作，覆盖父类中的 `performOk` 方法，可以添加“应用”按钮的操作。当然还可以覆盖父类中其他方法，对页面作进一步的设置，这部分内容将在 18.6 节详细讲述。
- ❑ 为了使读者对所有首选项页面的父类 `PreferencePage` 类进一步的了解，表 18.1 列举了该类中的常用方法，这些方法有些可以覆盖，有些可以直接在子类中调用。

表 18.1 `PreferencePage` 类中的主要方法及其说明

方法名称	说 明
<code>void applyData(Object data)</code>	应用数据，一般可以覆盖该方法
<code>void applyDialogFont(Composite composite)</code>	设置对话框的字体
<code>void contributeButtons(Composite parent)</code>	添加按钮时覆盖该方法
<code>abstract Control createContents(Composite parent)</code>	创建页面的各种控件，需子类实现
<code>void createControl(Composite parent)</code>	创建这个页面的控件，包括标题、按钮等，一般不需要覆盖该方法
<code>Label createDescriptionLabel(Composite parent)</code>	创建页面的标签区域
<code>Composite createNoteComposite(Font font, Composite composite, String title, String message)</code>	创建页面的注释区域
<code>Button getApplyButton()</code>	获得“应用”按钮对象
<code>Button getDefaultsButton()</code>	获得“默认”按钮对象
<code>IPreferencePageContainer getContainer()</code>	获得放置该页面的容器对象
<code>IPreferenceStore getPreferenceStore()</code>	获得保存选项值对象
<code>boolean isValid()</code>	该页面是否有效
<code>void noDefaultAndApplyButton()</code>	不显示“默认”和“应用”按钮
<code>boolean okToLeave()</code>	是否可以离开该页面
<code>void performApply()</code>	单击“应用”按钮调用 <code>performOk()</code> 方法

续表

方 法 名 称	说 明
boolean performCancel()	单击“取消”按钮调用该方法
void performDefaults()	单击“恢复缺省值”按钮调用该方法
boolean performOk()	单击“应用”按钮调用该方法
void performHelp()	单击“帮助”按钮调用该方法
void setContainer(IPreferencePageContainer container)	设置该页面所载的容器对象
void setPreferenceStore(IPreferenceStore store)	设置此页面所保存的选项值 IPreferenceStore 对象
void setTitle(String title)	设置页面标题
void setValid(boolean b)	设置该页面是否有效

18.3.2 创建首选项页面所对应的节点

有了页面，下面来看一下如何将树型导航栏中的一个节点关联到这个页面中。节点所对应的类为 PreferenceNode 类。该类的构造方法有 3 个，下面分别介绍。

1. PreferenceNode(String id)

其中，id 为标识该节点的字符，使用该构造方法需要使用 setPage 方法才可以将一个节点对象和一个页面对象关联起来，例如以下代码：

```
SystemSettingPage system = new SystemSettingPage();
PreferenceNode node = new PreferenceNode("System");
node.setPage( system );
```

2. PreferenceNode(String id, IPreferencePage preferencePage)

其中，id 为该节点的唯一标识符，PreferencePage 为该节点所对应的页面，使用该构造方法可以达到与以上代码相同的效果。

```
SystemSettingPage system = new SystemSettingPage();
PreferenceNode node = new PreferenceNode("System", system);
```

3. PreferenceNode(String id, String label, ImageDescriptor image, String className)

其中，id 为该节点的唯一标识符，label 为节点显示的名称，image 为节点所显示图标，className 为该节点所对应页面对象类的名称，注意是全称，包括所在的包名，例如 SystemSettingPage 类的全称是 com.fengmanfei.ch18.SystemSettingPage。还可以另一种方式获得类的全称的方法是 SystemSettingPage.class.getName()，也可以动态地获得该类的全称。该构造方法是最常用的构造方法，如下代码显示了如何使用此构造方法：

```
PreferenceNode node = new PreferenceNode("System",//唯一标识符
    "系统设置",//节点的名称
    ImageDescriptor.createFromFile(Constants.class, "tree_mode.gif"),//节点的图标
    SystemSettingPage.class.getName());//所对应的页面类的全称
```

一个 PreferenceNode 节点对象不仅保存了该节点所关联的首选项页面，还保存了该节点的其他信息，包括子节点等。表 18.2 列举了 PreferenceNode 类其他的一些常用方法。

表 18.2 PreferenceNode 类中的主要方法及其说明

方 法 名 称	说 明
void add(IPreferenceNode node)	添加该节点的子节点
void createPage()	创建节点所关联的首选项页面
IPreferenceNode findSubNode(String id)	查找该节点下是否有指定标识的节点对象
String getId()	获得该节点的标识符
ImageDescriptor getImageDescriptor()	获得该节点的图标对象
Image getLabelImage()	获得该节点标签的图标对象
String getLabelText()	获得该节点的名称
IPreferencePage getPage()	获得该节点所关联的页面
IPreferenceNode[] getSubNodes()	获得该节点所有的子节点对象
IPreferenceNode remove(String id)	移除指定标识的子节点对象
boolean remove(IPreferenceNode node)	移除指定子节点对象
void setPage(IPreferencePage newPage)	设置该节点所关联的首选项页面

18.3.3 显示首选项对话框

有了页面和所关联的节点对象，接下来就要将节点显示到界面上，这时需要使用 PreferenceManager 类，PreferenceManager 负责管理所有的节点。该类有两个构造方法，下面分别进行介绍。

1. PreferenceManager()

无参的构造方法。使用默认的“.”字符作为分隔符使用该构造方法，创建对象的代码如下：

```
PreferenceManager manager = new PreferenceManager();
```

2. PreferenceManager(char separatorChar)

其中 separatorChar 为路径分隔符。将一个节点添加到 PreferenceManager 对象使用以下两种方法：

(1) addToRoot(IPreferenceNode node)：添加到根节点中，例如以下代码所示为将一个节点添加到根节点中：

```
//创建 PreferenceManager 对象
PreferenceManager manager = new PreferenceManager();
//创建节点对象
PreferenceNode node = new PreferenceNode("System",
    "系统设置",
    ImageDescriptor.createFromFile(Constants.class, "tree_mode.gif"),
    SystemSettingPage.class.getName());
//添加到根节点上
manager.addToRoot(node);
```

(2) addTo(String path, IPreferenceNode node)：将一个节点添加到指定路径的节点上，

其中 `path` 为指定的路径, `node` 为要添加的节点。

下面再来看一下 `PreferenceManager` 类所常用的方法, 如表 18.3 所示。

表 18.3 `PreferenceManager` 类中的主要方法及其说明

方法名称	说 明
<code>boolean addTo(String path, IPreferenceNode node)</code>	将指定的节点附加到指定的路径上
<code>void addToRoot(IPreferenceNode node)</code>	将指定的节点附加到根节点上
<code>IPreferenceNode find(String path)</code>	根据指定路径查找指定的节点
<code>IPreferenceNode find(String path, IPreferenceNode top)</code>	查找指定节点下的路径所在的节点
<code>List getElements(int order)</code>	返回所有的节点, 其中 <code>order</code> 的值可以使用 <code>PRE_ORDER</code> 和 <code>POST_ORDER</code> , <code>PRE_ORDER</code> 表示首先对根节点进行排序, <code>POST_ORDER</code> 表示首先对子节点进行排序
<code>IPreferenceNode remove(String path)</code>	移除指定路径的节点
<code>boolean remove(IPreferenceNode node)</code>	移除指定的节点
<code>void removeAll()</code>	移除所有节点

最后, 将所有的这些对象串联起来, 使用 `PreferenceDialog` 窗口将树型导航栏和选项页面显示出来。 `PreferenceDialog` 的构造方法是:

```
PreferenceDialog(Shell parentShell, PreferenceManager manager)
```

编写一个测试程序, 将调用“首选项”对话框。该测试程序代码如下:

PreferenceTest.java

```
package com.fengmanfei.ch18;
import java.io.IOException;
import org.eclipse.jface.preference.PreferenceDialog;
import org.eclipse.jface.preference.PreferenceManager;
import org.eclipse.jface.preference.PreferenceNode;
import org.eclipse.jface.preference.PreferenceStore;
import org.eclipse.swt.widgets.Display;

public class PreferenceTest {

    public static void main(String[] args) {
        Display display = new Display();

        //创建一个 PreferenceManager 对象
        PreferenceManager manager = new PreferenceManager();
        //创建一个节点对象
        PreferenceNode nodeOne = new PreferenceNode("System", "系统设置", null, System
        SettingPage.class.getName());
        //将该节点添加到根节点中
        manager.addToRoot(nodeOne);
        //定义一个“首选项”对话框, 并将 manager 作为参数传入
```

```

        PreferenceDialog dlg = new PreferenceDialog(null, manager);
        //创建保存选项设置值对象
        PreferenceStore preferenceStore = new PreferenceStore("F:\\myPreference.
properties");
        try {
            //装载该文件中的设置值
            preferenceStore.load();
            //将该设置赋值给该对话框
            dlg.setPreferenceStore(preferenceStore);
            //打开对话框
            dlg.open();
            //最后关闭对话框后，保存当前设置
            preferenceStore.save();
        } catch (IOException e) {
            e.printStackTrace();
        }
        display.dispose();
    }
}

```

这样，就创建了只有一页首选项页面的对话框。通常首选项页面一般不会只有一个页面，会显示多个页面，这就涉及如何创建树型菜单的知识。

18.4 创建树型的导航菜单

要创建树型菜单，有两种方法，一种是在创建节点时就已经创建了该节点中的子节点，这种情况下使用 `PreferenceNode` 对象的 `add` 方法为节点添加子节点。另一种方法是通过 `PreferenceManager` 对象的 `addTo` 方法为指定路径下添加节点。

下面以如图 18.4 所示的结构为例说明如何使用两种方法，以达到同样的效果。

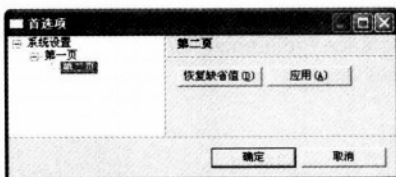


图 18.4 树型导航栏

为了创建如图 18.4 所示的树型导航栏，首先要创建 `PageOne` 和 `PageTwo` 两个选项页面，暂时这里不创建任何控件。`PageOne` 选项页的代码如下所示，`PageTwo` 的代码与 `PageOne` 的代码相同，只是类名不同。

PageOne.java

```
package com.fengmanfei.ch18;
```

```
import org.eclipse.jface.preference.PreferencePage;
import org.eclipse.swt.widgets.*;
```

```
public class PageOne extends PreferencePage{
    protected Control createContents(Composite parent) {
        return parent;
    }
}
```

18.4.1 第一种方法

第一种方法创建树型菜单，修改 PreferenceTest 的代码如下：

```
//创建一个 PreferenceManager 对象
PreferenceManager manager = new PreferenceManager();
//创建一个节点对象
PreferenceNode nodeOne = new PreferenceNode("System", "系统设置", null, System Setting
Page.class.getName());
//将该节点添加到根节点中
manager.addToRoot(nodeOne);
//创建两个节点对象
PreferenceNode one = new PreferenceNode("one", "第一页", null, PageOne.class.getName());
PreferenceNode two = new PreferenceNode("two", "第二页", null, PageTwo.class.getName());
//第二页节点为第一页节点的子节点
one.add( two );
//第一页节点为系统设置节点的子节点
nodeOne.add(one);
```

18.4.2 第二种方法

第二种方法创建树型菜单，修改 PreferenceTest 的代码如下：

```
//.....以上代码相同，省略
PreferenceNode one = new PreferenceNode("one", "第一页", null, PageOne.class.getName());
PreferenceNode two = new PreferenceNode("two", "第二页", null, PageTwo.class.getName());
//将第一页节点附加到 System 路径下
manager.addTo("System",one);
//将第二页节点附加到 System.one 路径下
manager.addTo("System.one",two);
```

通过这两种方法创建的树型导航的效果是一样的。两种方法都各有优点，在创建某一个节点和其子节点时，通常使用第一种方法。在修改子节点时，通常使用第二种方法。

18.5 首选项的选项设置

对于首选项的设置，可以使用 SWT 基本的控件就可以实现，但为了方便创建设置的选

项, JFace 提供了常用的一些设置选项值的字段对象。

18.5.1 字段编辑器概述

字段编辑器类 `FieldEditor` 类是一个抽象类, 它的子类的继承关系如图 18.5 所示。创建字段编辑器要实现其方法的子类。

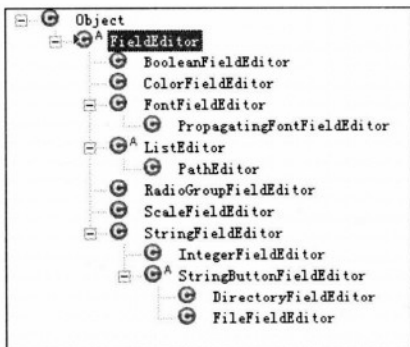


图 18.5 字段编辑器的继承关系图

使用字段编辑器的好处是, 不必考虑对选项的保存, 只需要创建使用的编辑器对象即可。下面就来看一下如何使用编辑器字段对象。

18.5.2 使用字段编辑器基本步骤

下面以创建一个 `StringFieldEditor` 字符型编辑的程序来说明如何使用字段编辑器。如图 18.6 所示, 这是一个创建了 `StringFieldEditor` 对象的效果图。

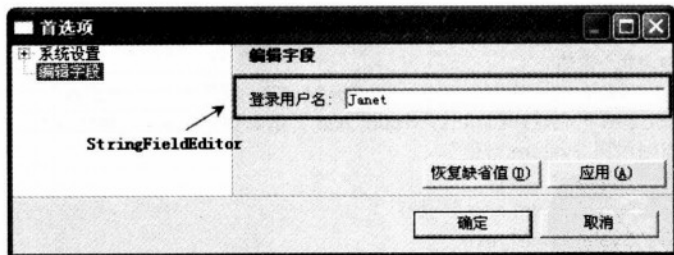


图 18.6 字符型字段编辑器

创建这样一个字符型编辑器的代码非常简单, 如下:

```
package com.fengmanfei.ch18;

import org.eclipse.jface.preference.FieldEditorPreferencePage;
import org.eclipse.jface.preference.StringFieldEditor;
```



```

public class FieldEditorPage extends FieldEditorPreferencePage {
    public FieldEditorPage() {
        super(GRID); //页面的样式, 还可以使用 FLAT 常量
    }
    //该方法为实现父类中的抽象方法, 在该方法中添加所需的编辑器对象
    protected void createFieldEditors() {
        //创建一个字符型编辑器对象
        StringFieldEditor userName = new StringFieldEditor(
            Constants.USER_NAME, //选项的 key 值
            "登录用户名:", //显示的标签名
            getFieldEditorParent()); //该字段编辑器的父类面板
        //调用父类方法, 将该方法添加到该页中
        addField(userName);
    }
}

```

通过以上程序代码, 总结出使用字段编辑器应注意以下方面:

- ❑ 要使用字段编辑器, 创建首选项页面时要继承自 `FieldEditorPreferencePage` 类, 而不是继承 `PreferencePage` 类, 而且要实现 `createFieldEditors` 抽象方法, 在该方法中创建各种编辑器对象。
- ❑ 创建编辑器对象时, 至少要有 3 个参数, 选项的 key 值、显示的标签名和该字段编辑器的父类面板。其中选项的 key 值, 即为保存为文件中的 key 值, 编辑对象将会自动显示文件中所设置的值。
- ❑ 创建完编辑器对象后, 要调用父类的 `addField` 方法, 将该编辑器对象添加到页面中。

现在, 请读者将使用编辑器创建的首选项页面与 18.3.1 节所创建的首选项页面的代码作一下比较。很明显, 同样是设置选项值, 使用编辑器极大地简化了代码的开发量。另外也可以看到, 使用编辑器时, 也不需要编写单击“恢复缺省值”和“应用”按钮的操作, 这是因为 `FieldEditorPreferencePage` 是 `PreferencePage` 的子类, 如图 18.7 所示。

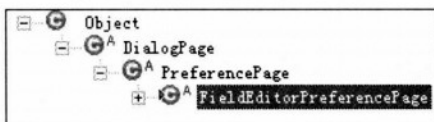


图 18.7 `FieldEditorPreferencePage` 类继承关系

在 `FieldEditorPreferencePage` 类中已经覆盖了 `PreferencePage` 类中的 `performOk` 方法, 并且循环所有的字段编辑器对象, 保存到存储设置的对象中。为了使读者更清楚地明白该方法的作用, 以下是 `FieldEditorPreferencePage` 类中 `performOk` 方法的源代码:

```

public boolean performOk() {
    if (fields != null) {
        Iterator e = fields.iterator();
        while (e.hasNext()) {
            FieldEditor pe = (FieldEditor) e.next();
            pe.store();
            pe.setPresentsDefaultValue(false);
        }
    }
}

```

```

    }
}
return true;
}

```

所以在使用字段编辑器对象时，就不需要考虑存储的问题了。所做的只是赋给字段编辑器所使用的 key，其他的就交给字段编辑器自己处理了。

18.5.3 布尔型字段编辑器 (BooleanFieldEditor)

布尔型字段编辑器是用于设置 true 和 false 值的编辑器，如图 18.8 所示为一个布尔型字段编辑器的效果。

创建这样的布尔型字段编辑器代码如下：

图 18.8 布尔型字段编辑器效果图

```

BooleanFieldEditor bfe = new BooleanFieldEditor("show", "Boolean", getFieldEditorParent());
addField(bfe);

```

18.5.4 颜色字段编辑器 (ColorFieldEditor)

颜色字段编辑器用于设置颜色值，当单击按钮后可以弹出颜色选择对话框，并且当选定中某一个颜色时，将该颜色显示到按钮上。如图 18.9 所示为一个颜色字段编辑器的效果。



图 18.9 颜色字段编辑器效果图

创建这样的颜色字段编辑器代码如下：

```

ColorFieldEditor cfe = new ColorFieldEditor("color", "Color", getFieldEditorParent());
addField(cfe);

```

18.5.5 字体字段编辑器 (FontFieldEditor)

字体字段编辑器用于设置字体值，当单击“更改”按钮后可以弹出字体选择对话框，并且当设置完字体后，将当前设置的字体信息显示在界面上。如图 18.10 所示为一个字体字段编辑器的效果。



图 18.10 字体字段编辑器效果图

创建这样的字体字段编辑器代码如下：

```

FontFieldEditor ffe = new FontFieldEditor("font", "Font", getFieldEditorParent());
addField(fffe);

```

18.5.6 路径列表字段编辑器 (PathEditor)

路径列表字段编辑器是一个列表编辑器，单击“新建”按钮，弹出路径选择对话框，确定后将在列表中添加一个路径。也可以删除一个路径值，或者改变列表中的顺序。如图 18.11 所示为一个路径列表字段编辑器的效果。

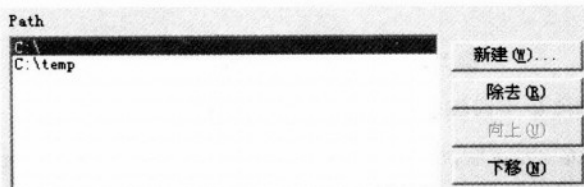


图 18.11 路径列表字段编辑器的效果图

创建这样的路径列表字段编辑器代码如下：

```
PathEditor pfe = new PathEditor("path", "Path", "请选择所选的路径", getFieldEditorParent());
addField(pfe);
```

其中，第三个参数为弹出路径选择对话框时，对话框提示信息的字符。

18.5.7 单选分组字段编辑器 (RadioGroupFieldEditor)

单选分组字段编辑器将一组单选按钮以一个分组的方式显示出来，并且可以设定每行所显示单选按钮的个数。如图 18.12 所示为单选分组字段编辑器的效果图。

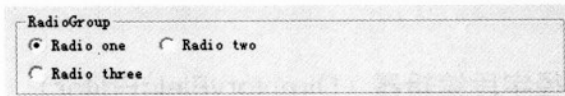


图 18.12 单选分组字段编辑器的效果图

创建这样一个单选分组字段编辑器的代码如下：

```
RadioGroupFieldEditor rgfe = new RadioGroupFieldEditor(
    "group", //选项的 key
    "RadioGroup", //分组框显示的文本
    2, //一行显示的单选按钮个数
    new String[] { {"Radio one", "one"}, {"Radio two", "two"}, {"Radio three", "three"} }, //单选
    按钮的标签和值
    getFieldEditorParent(), //父类的面板
    true); //true 表示使用分组面板，false 将使用普通的面板
addField(rgfe);
```

18.5.8 刻度条字段编辑器 (ScaleFieldEditor)

刻度条字段编辑器是带有标签和刻度条的编辑器对象。如图 18.13 所示为刻度条字段编辑器的效果图。

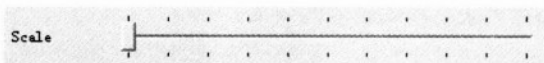


图 18.13 刻度条字段编辑器的效果图

创建这样一个刻度条字段编辑器的代码如下：

```
ScaleFieldEditor sfe = new ScaleFieldEditor("scale", "Scale", getFieldEditorParent(), 0, 100, 5, 10);  
addField(sfe);
```

18.5.9 整数型字段编辑器 (IntegerFieldEditor)

整数型字段编辑器是只可以输入整数的编辑器，当输入的不是整数型字符时，将显示错误信息。如图 18.14 所示为整数型字段编辑器以及在提示错误信息的情况下的效果图。

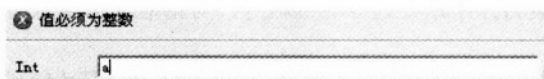


图 18.14 整数型字段编辑器效果图

创建这样一个整数型字段编辑器的代码如下：

```
IntegerFieldEditor ife = new IntegerFieldEditor("int", "Int", getFieldEditorParent());  
addField(ife);
```

18.5.10 选择路径字段编辑器 (DirectoryFieldEditor)

选择路径字段编辑器可以选择一个路径，单击“浏览”按钮，将可以弹出目录选择对话框，确定后，将保存该路径的地址。如图 18.15 所示为选择路径字段编辑器的效果图。



图 18.15 选择路径字段编辑器效果图

创建这样一个选择路径字段编辑器的代码如下：

```
DirectoryFieldEditor dfe = new DirectoryFieldEditor("directory", "Directory", getFieldEditorParent());  
addField(dfe);
```

18.5.11 选择文件字段编辑器 (FileFieldEditor)

选择文件字段编辑器与选择路径字段编辑器类似，都有一个“浏览”按钮，但所不同

的是单击“浏览”按钮后，弹出的是文件选择对话框，而不是目录选择对话框。如图 18.16 所示为选择文件字段编辑器的效果图。

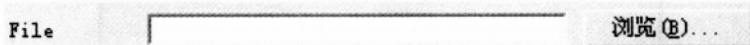


图 18.16 选择文件字段编辑器效果图

创建这样一个刻度条字段编辑器的代码如下：

```
FileFieldEditor filefe = new FileFieldEditor("file", "File", getFieldEditorParent());
addField(filefe);
```

最后，将所有的字段编辑器显示到同一个页面上，程序运行后的效果如图 18.17 所示。

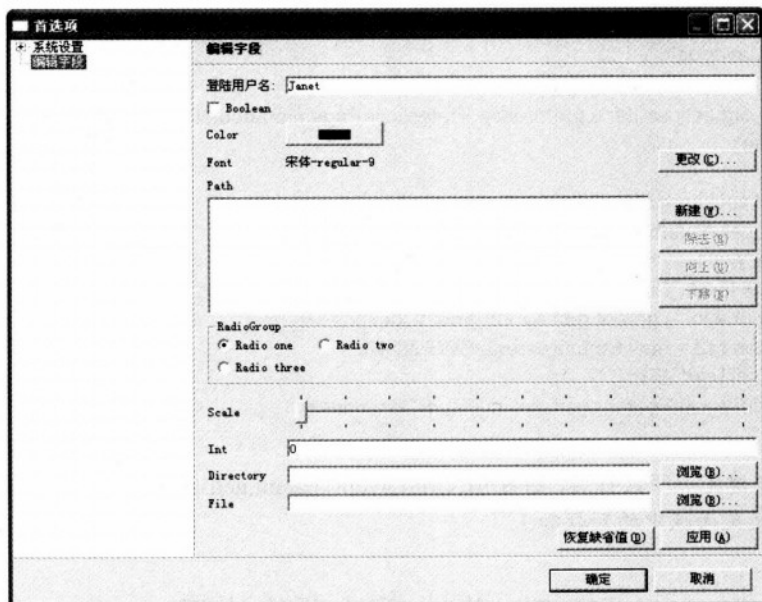


图 18.17 所有的字段编辑器效果图

18.6 自定义首选项页面

每个首选项页面都有一个“恢复缺省值”按钮和“应用”按钮。若要想自定义首选项页面上的按钮，需要覆盖父类中的 `contributeButtons` 方法，下面回到 18.3.1 节中的 `SystemSettingPage` 类，在该页上添加两个按钮后的效果如图 18.18 所示。

添加自定义按钮的代码如下：

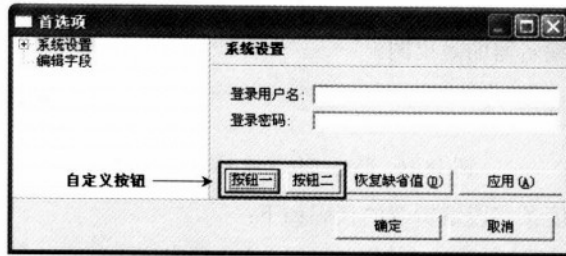


图 18.18 自定义按钮效果

SystemSettingPage.java

```

/*
 * (非 Javadoc)
 *
 * @see org.eclipse.jface.preference.PreferencePage#contributeButtons(org.eclipse.swt.widgets.
 * Composite)
 */
protected void contributeButtons(Composite parent) {
    // super.contributeButtons(parent);
    Button bt1 = new Button(parent, SWT.NONE);
    bt1.setText("按钮一");
    ((GridLayout) parent.getLayout()).numColumns++;
    Button bt2 = new Button(parent, SWT.NONE);
    bt2.setText("按钮二");
    ((GridLayout) parent.getLayout()).numColumns++;
}

```

⚠注意: 在创建完一个按钮后, 要调用 `((GridLayout) parent.getLayout()).numColumns++` 方法, 将父类面板中的列数加 1。

18.7 首选项的事件处理

与向导对话框类似, 当切换首选项页面时也会触发 `PageChangedEvent` 事件。为首选项对话框注册页面切换事件使用的是 `addPageChangedListener` 方法, 代码如下:

```

PreferenceDialog dlg = new PreferenceDialog(null, manager);
//注册页面切换事件
dlg.addPageChangedListener( new IPageChangedListener(){
    //当页面切换时
    public void pageChanged(PageChangedEvent event) {
        //获得当前页面
        IPreferencePage page = (IPreferencePage)event.getSelectedPage();
        //输出当前页面的标题
        System.out.println(page.getTitle());
    }
}

```

```
}  
});
```

移除该事件的方法是 `removePageChangedListener`。

18.8 本章小结

本章学习了有关首选项的内容，首选项在任何应用程序中是必不可少的，可以按照用户的喜好来设置。JFace 提供了一套非常便捷的创建首选项的类库，包括页面的显示、树型导航栏等，也提供了一些专门用于设置首选项的字段编辑器。另外，对首选项设置的保存和获取也非常容易。Eclipse 工作平台完全是基于 JFace 首选项的。所以，读者如果进行 Eclipse 插件开发，就明白其中的原理了。




第 19 章 MVC 的表格、树和列表

本章将详细介绍 MVC 模式的表格、树和列表控件。在之前的 SWT 中已经学习过表格 (Table)、树 (Tree) 和列表 (List)，但使用这些控件时数据与视图高度耦合，不利于实际项目中将业务逻辑与视图相分离。JFace 改进了这些控件，实现了 MVC 的模式 (Model-View-Controller) 的表格、树和列表，使之功能更加强大。

19.1 MVC 概述

MVC 是模型-视图-控制器 (Model-View-Controller) 设计模式的简称。MVC 设计模式将数据与视图分开，通过控制器来控制与数据交互。JFace 中有关实现 MVC 模式的控件都在 `org.eclipse.jface.viewers` 包下。所有的 MVC 的控件都继承自 `View` 类，该类是一个抽象类，不能直接创建对象，需使用其实现的子类。常使用的类如下所示：

- ❑ `ListViewer` 类：列表控件。
- ❑ `TreeViewer` 类：树控件。
- ❑ `TableViewer` 类：表格控件。
- ❑ `TextViewer`、`SourceViewer` 和 `ProjectionViewer` 为编辑文本的控件，有关 JFace 中文本处理的操作将在第 21 章中讲述。

如图 19.1 所示为这些类的继承关系示意图，其中带  图标类为抽象类，不能直接创建对象。

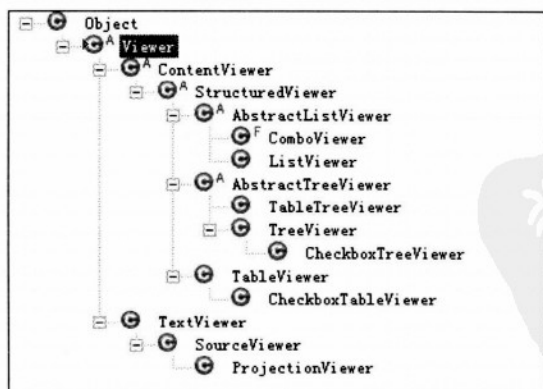


图 19.1 JFace 的 Viewer 类继承关系图

19.2 表格 (TableViewer)

首先以一个示例程序来学习如何使用 MVC 的表格。如图 19.2 所示为该表格程序运行后的效果图。该表格运行后效果与之前使用 SWT 的表格没有什么不同，但编写的代码要简单得多。

下面就学习如何一步步地创建这样的表格程序。

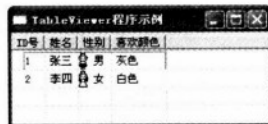


图 19.2 TableViewer 表格程序示例

19.2.1 创建表格控制器 (Controller)

先来看一下如何创建图 19.2 所示的表格的代码，然后再进行详细的分析。该表格程序实现的具体代码如下：

TableWindow.java

```
package com.fengmanfei.ch19;
import java.util.ArrayList;
import java.util.List;
import org.eclipse.jface.viewers.*;
import org.eclipse.jface.window.ApplicationWindow;
import org.eclipse.swt.SWT;
import org.eclipse.swt.graphics.Image;
import org.eclipse.swt.widgets.*;
import com.fengmanfei.util.ImageFactory;

public class TableWindow extends ApplicationWindow {

    private TableViewer table;//声明一个表格对象
    private List persons;//表格所存储的数据
    //表格中列的索引号
    public static final int ID = 0;
    public static final int NAME = 1;
    public static final int GENDER = 2;
    public static final int COLOR = 3;
    //表格中的列名称
    public static final String[] COLUMN_NAME = {"ID 号", "姓名", "性别", "喜欢颜色"};
    public TableWindow() {
        super(null);
        initPersons();//初始化表格数据
    }
    //初始化表格数据，通常是从数据库中读出
    private void initPersons() {
        persons = new ArrayList();
        persons.add( new PersonEO(1,"张三","男","灰色"));
    }
}
```

```
        persons.add( new PersonEO(2,"李四","女","白色"));
    }
    //设置窗口的标题和大小
    protected void configureShell(Shell shell) {
        super.configureShell(shell);
        shell.setSize(300, 200);
        shell.setText("TableViewer 程序示例");
    }
    //创建窗口的控件
    protected Control createContents(Composite parent) {
        //创建 TableViewer 对象
        table = new TableViewer(parent, SWT.FULL_SELECTION);
        //创建表头
        for ( int i =0; i<COLUMN_NAME.length;i++){
            new TableColumn(table.getTable(), SWT.LEFT).setText(COLUMN_NAME[i]);
            table.getTable().getColumn(i).pack();
        }
        //设置表头和表格线可见
        table.getTable().setHeaderVisible(true);
        table.getTable().setLinesVisible( true );
        //设置数据
        table.setContentProvider(new MyContentProvider());
        //设置视图
        table.setLabelProvider( new MyLabelProvider());
        //设置表格数据对象，该方法非常重要，是表格数据入口
        table.setInput(persons);

        return parent;
    }

    public static void main(String[] args) {
        TableWindow test = new TableWindow();
        test.setBlockOnOpen(true);
        test.open();
        Display.getCurrent().dispose();
    }
}
```

从以上程序的代码中总结出使用 TableViewer 对象时应注意以下问题：

1. 创建 TableViewer 的构造方法

TableViewer 类的构造方法有 3 个，分别是：

- ☐ TableViewer(Composite parent)：使用默认的风格。
- ☐ TableViewer(Composite parent, int style)：可设置表格的风格，风格常量可以使用 Table 类使用的风格常量，本例中使用的就是这个构造方法。
- ☐ TableViewer(Table table)：可以将一个 SWT 表格对象作为构造参数。

2. TableViewer 中的 Table 对象

TableViewer 实际上封装了 SWT 的 Table 对象, 通过 `table.getTable()` 可以获得该 Table 对象。表头的设置和显示与之前学习的 SWT 的表格的方法相同。

3. 显示表格数据的一般步骤

虽然 TableViewer 内部使用 Table 对象, 但创建表格数据的代码中并不用创建 TableItem 来组织数据, 这是因为 TableViewer 将组织和显示的数据交给了两个接口对象 IContent Provider 和 ITableLabelProvider。

IContentProvider 负责将 setInput 方法传入的对象转化为表格所需要的对象。而 ITableLabelProvider 负责处理如何显示数据。这两个接口的使用在下面将详细介绍。总之, 要组织表格数据并显示出数据, 必须使用以下 3 种方法:

```
//设置数据
table.setContentProvider(new MyContentProvider());
//设置视图
table.setLabelProvider( new MyLabelProvider());
//设置表格数据对象, 该方法非常重要, 是表格数据入口
table.setInput(persons);
```

19.2.2 创建表格模型 (Model)

表格中数据组织与两个方法有关, 一个是 setInput 方法, 一个是 setContentProvider 方法。先来看一下 setInput 方法。

```
void setInput(Object input)
```

该方法为表格数据最初的入口点, 任何对象都可以设置为表格数据, 因为该方法所设定的对象为 Object 对象, 所有的类都是 Object 子类。但通常使用集合类来组织数据, 该程序中使用的是 List 对象, List 对象存放了两个 PersonEO 实体对象。

在实际的项目开发中, 一般都会使用实体对象 (Entity Object) 来装载数据库中的数据, 可以这样理解实体对象, 一个实体对象代表了数据库中表的条记录, 而该实体对象的属性就是表的列属性。该实体对象除了有属性外还有设置和获取属性的方法。

例如, 本例中使用 PersonEO 实体对象, 该类实现的具体代码如下:

PersonEO.java

```
package com.fengmanfei.ch19;

public class PersonEO {
    private int ID;
    private String name;
    private String gender;
    private String color;
    public PersonEO (){}
    public PersonEO ( int id , String name , String gender, String color){
```

```
        this.ID = id;
        this.name = name;
        this.gender = gender;
        this.color = color;
    }
    public String getColor() {
        return color;
    }
    public void setColor(String color) {
        this.color = color;
    }
    public String getGender() {
        return gender;
    }
    public void setGender(String gender) {
        this.gender = gender;
    }
    public int getID() {
        return ID;
    }
    public void setID(int id) {
        ID = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

从该实体类中可以看出，该类的属性对应表中的一列。除了列属性外，还有设置和获取属性的方法。

将实体对象装载到 List 对象中，然后通过 setInput 方法设置给 TableViewer 对象来使用。如图 19.3 所示，表示了 List 对象与表格中数据的关系。

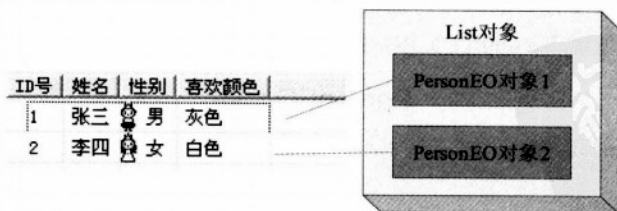


图 19.3 List 对象与实体对象的关系图

虽然有了初始的数据，TableViewer 还不知道怎样将数据分配到对应的表格单元中，这时就需要使用 setContentProvider 方法为表格创建一个规则，使初始化的数据对象转化为表格中的数据。

```
void setContentProvider(IContentProvider provider)
```

IContentProvider 是一个接口，而在表格使用时实现 IStructuredContentProvider 这个接口，因为 IStructuredContentProvider 继承自 IContentProvider。例如，本例中使用了一个内部类 MyContentProvider，该类实现了 IStructuredContentProvider 接口，具体的代码如下：

```
public class MyContentProvider implements IStructuredContentProvider {
    //将初始化数据的入口对象转换成表格使用的数组对象
    public Object[] getElements(Object inputElement) {
        return ((List) inputElement).toArray();
    }
    //释放该对象时调用的方法
    public void dispose() {
    }
    //当表格中的数据改变时调用该方法
    public void inputChanged(Viewer viewer, Object oldInput, Object newInput) {
    }
}
```

实现 IStructuredContentProvider 接口必须实现 3 个方法，其中最重要的是 getElements 方法。该方法主要是将 setInput 设置的对象转化成表格所使用的数组对象。另外还应该注意，接口中使用的参数许多都是 Object，这就提供了很大的灵活性，但在运行期间容易产生类型转换异常，所以在使用时一定要小心。如果在不知道是何类型的情况下可以使用以下代码进行类型检查：

```
public Object[] getElements(Object inputElement) {
    if (inputElement instanceof List)
        return ((List) inputElement).toArray();
    return null;
}
```

另外两个方法，暂时空实现，什么都不做。

19.2.3 创建组织表格视图 (View)

有了数据，还要让表格知道如何显示数据，这时就需要使用 setLabelProvider (IbaseLabel Provider labelProvider)方法。与 setContentProvider 方法类似，IBaseLabelProvider 也是一个接口，在表格中可以通过实现 ITableLabelProvider 接口，这个接口是 IBaseLabelProvider 的子类。同样，本例中是使用一个内部类实现该接口的，具体代码如下：

```
public class MyLabelProvider implements ITableLabelProvider {

    //设置每个单元格所显示的图标
    public Image getColumnImage(Object element, int columnIndex) {
        //如果是性别所在的列
        if (columnIndex == GENDER){
            //类型转换，element 代表表格中的一行
        }
    }
}
```

```

        PersonEO person = (PersonEO)element;
        //根据不同的值设置不同的图标
        if ( person.getGender().equals("男"))
            return ImageFactory.loadImage( Display.getCurrent(),ImageFactory.I CON_BOY);
        else if ( person.getGender().equals("女"))
            return ImageFactory.loadImage( Display.getCurrent(),ImageFactory.I CON_GIRL);
        }
        return null;
    }
    //设置每个单元格所显示的文字
    public String getColumnText(Object element, int columnIndex) {
        //类型转换, element 代表表格中的一行
        PersonEO person = (PersonEO)element;
        if ( columnIndex == ID)//如果是第一列
            return person.getID()+" ";
        else if ( columnIndex == NAME)//如果是第二列
            return person.getName()+" ";
        else if ( columnIndex == GENDER)//如果是第三列
            return person.getGender()+" ";
        else if ( columnIndex == COLOR)//如果是第四列
            return person.getColor()+" ";
        return " ";
    }
    //释放对象时释放图像资源
    public void dispose() {
        ImageFactory.dispose();
    }
    //以下方法为空实现
    public void addListener(ILabelProviderListener listener) {
    }
    public boolean isLabelProperty(Object element, String property) {
        return false;
    }
    public void removeListener(ILabelProviderListener listener) {
    }
}

```

在实现该接口中, 最重要的是要实现 `getColumnImage` 和 `getColumnText` 方法。一个是设置单元格的图标, 一个是设置单元格显示的文本。注意, 这里使用的参数也为 `Object` 类型, 必要时需要进行类型判断。

综上所述, 通过以上这 3 个步骤, 一个表格就基本实现了, 与之前只使用 SWT 中的表格不同, 不用再创建 `TableItem` 来创建数据了, 极大地简化了代码。

19.2.4 添加和删除数据

下面为表格增加添加数据和删除数据的功能, 首先要为表格添加一个右键弹出的上下文菜单。菜单上有两个菜单项——“添加”和“删除”菜单项, 单击“添加”后, 向表中

增加一条记录，单击“删除”，则将表中当前选中的记录删除。添加完上下文菜单后程序运行的效果如图 19.4 所示。

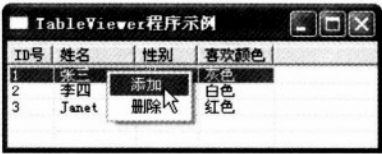


图 19.4 添加上下文菜单后的表格

首先要创建所使用的菜单，之前在第 15 章中介绍过 `MenuManager` 对象和 `Action` 对象来创建菜单，这里也采用该方法。编写 `createContextMenu()` 方法专门用于创建菜单。该方法的具体代码如下：

```
//创建上下文菜单
private void createContextMenu() {
    MenuManager menu = new MenuManager();
    menu.add( new AddAction() );
    menu.add( new DelAction() );
    //获得一个 Menu 对象
    Menu m = menu.createContextMenu(getShell());
    //将该对象设置为表格的菜单
    table.getTable().setMenu( m );
}
```

另外两个菜单项是通过 `AddAction` 对象和 `DelAction` 对象来实现对表格的操作的。这两个类也通过内部类来实现，具体代码如下：

AddAction

```
public class AddAction extends Action{
    public AddAction(){
        setText("新建");
    }
    public void run() {
        //新建一个实体对象
        PersonEO person = new PersonEO();
        person.setID( table.getTable().getItemCount()+1);
        person.setName("Janet");
        person.setGender("女");
        person.setColor("红色");
        //添加一行数据
        table.add( person );
    }
}
```


DelAction

```
public class DelAction extends Action{
    public DelAction(){
        setText("删除");
    }

    public void run() {
        //获得当前所选中的行
        StructuredSelection selection = (StructuredSelection) table.getSelection();
        //注意，同时选中的可能是多行，这时返回的是数组对象
        //获得数组对象的一个元素，也就是只有选中一行的情况
        PersonEO p= (PersonEO)selection.getFirstElement();
        //从表中删除该行
        table.remove(p);
    }
}
```

从以上程序的代码中，总结一下对表格操作的一些常用的方法：

- ☐ 添加一行数据：add(Object element)。
- ☐ 添加多行数据：add(Object[] elements)。
- ☐ 清空指定表格索引一行的数据：clear(int index)。
- ☐ 在表格指定的索引行插入一行：insert(Object element, int position)。
- ☐ 删除指定的行：remove(Object element)。
- ☐ 删除指定的多行：remove(Object[] elements)。
- ☐ 替换指定行数据：replace(Object element, int index)。

但要注意，这些方法只是对表格显示的数据进行添加和删除，并未改变数据模型中提供的数据，在本例数据模型中的数据也就是 List 对象。也可以通过改变模型中的数据来实现对表格的添加和修改，但此时要使用 refresh()方法刷新表格。

19.2.5 增加表格排序功能

通常为了能够方便查看表格的内容，表格的表头一般都有排序功能，下面就来看一下如何为表格添加排序功能。要使表格进行排序，需要使用 setSorter(ViewerSorter sorter)方法为表格设置一个排序器。

在这里，专门写成一个类 TableSorter，该类继承自 ViewerSorter 并且覆盖了父类的 compare 方法。该类实现的具体代码如下：

TableSorter.java

```
package com.fengmanfei.ch19;

import org.eclipse.jface.viewers.Viewer;
import org.eclipse.jface.viewers.ViewerSorter;
```

```

public class TableSorter extends ViewerSorter {
    private static final int ASCENDING = 0;
    private static final int DESCENDING = 1;

    private int order;//判断是升序还是降序
    private int column;//判断排序的列

    public void doSort(int column) {
        // 如果是同一列，改变排列的顺序
        if (column == this.column) {
            order = 1 - order;
        } else { // 如果是另一列，则默认为升序排列
            this.column = column;
            order = ASCENDING;
        }
    }
    //覆盖父类的方法，返回比较结果有-1,0,1 3 种情况
    public int compare(Viewer viewer, Object e1, Object e2) {
        int result = 0;

        PersonEO p1 = (PersonEO) e1;
        PersonEO p2 = (PersonEO) e2;

        //默认是升序
        switch (column) {
            case TableWindow.ID:
                result = p1.getID() > p2.getID() ? 1 : -1;
                break;
            case TableWindow.NAME:
                result = collator.compare(p1.getName(), p2.getName());
                break;
            case TableWindow.GENDER:
                result = collator.compare(p1.getGender(), p2.getGender());
                break;
            case TableWindow.COLOR:
                result = collator.compare(p1.getColor(), p2.getColor());
                break;
        }
        //如果此时为降序
        if (order == DESCENDING)
            result = -result;
        return result;
    }
}

```

实现该排序器类的代码中，doSort 为单击表头时调用，而判断数据顺序的方法是覆盖父类中的 compare 方法。该方法的返回值用于判断数据的顺序。

有了排序器对象，下面就可以为表头注册单击表头事件，在初始化表头对象后，在程

序中增加以下代码：

```
//设置排序器
table.setSorter( new TableSorter());
//分别为表头的每一列注册事件
TableColumn column = table.getTable().getColumn(0);
column.addSelectionListener( new SelectionAdapter(){
    public void widgetSelected(SelectionEvent e) {
        //单击时，重新设置排序对象属性
        ((TableSorter)table.getSorter()).doSort(TableWindow.ID);
        //刷新表格数据
        table.refresh();
    }
});
column = table.getTable().getColumn(1);
column.addSelectionListener( new SelectionAdapter(){
    public void widgetSelected(SelectionEvent e) {
        ((TableSorter)table.getSorter()).doSort(TableWindow.NAME);
        table.refresh();
    }
});
column = table.getTable().getColumn(2);
column.addSelectionListener( new SelectionAdapter(){
    public void widgetSelected(SelectionEvent e) {
        ((TableSorter)table.getSorter()).doSort(TableWindow.GENDER);
        table.refresh();
    }
});
column = table.getTable().getColumn(3);
column.addSelectionListener( new SelectionAdapter(){
    public void widgetSelected(SelectionEvent e) {
        ((TableSorter)table.getSorter()).doSort(TableWindow.COLOR);
        table.refresh();
    }
});
```

做完以上修改后，再运行程序就可以看到能够排序的效果了。另外，当不需要排序器时，可使用 `setSorter(null)` 方法将排序器设置为 `null` 就可以了。

19.2.6 增加表格过滤功能

对表格中的数据通常会进行筛选过滤，与增加排序功能类似，可以通过 `addFilter(ViewerFilter filter)` 方法为表格设置过滤器。

下面要实现的功能是在上下文菜单中创建增加一个筛选菜单项，当选中此项时，将只显示所有性别为“女”的数据，如图 19.5 所示。

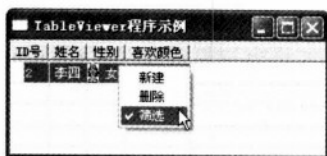


图 19.5 增加了筛选菜单项的效果示意图

首先要在创建菜单项的代码后增加如下代码：

```
menu.add( new FileterAction() );
```

然后再编写该菜单项的代码，FileterAction 为内部类，具体实现代码如下：

```
public class FileterAction extends Action{
    //声明一个 ViewerFilter 对象
    ViewerFilter femaleFilter;
    public FileterAction(){
        //设置标题，样式为复选框样式
        super("筛选",AS_CHECK_BOX);
        //内部类创建该对象
        femaleFilter = new ViewerFilter(){
            //需实现 ViewerFilter 类中的抽象方法
            public boolean select(Viewer viewer, Object parentElement, Object element) {
                PersonEO p = (PersonEO)element;
                return p.getGender().equals("女");
            }
        };
        //初始化时设置为取消选中状态
        this.setChecked(false);
    }

    public void run() {
        //如果此时选中，则为表格设置过滤器
        if (isChecked())
            table.addFilter(femaleFilter);
        else//否则，移除表格过滤器
            table.removeFilter(femaleFilter);
        //刷新表格数据
        table.refresh();
    }
}
```

当创建过滤器对象时，必须实现 select 方法，该方法返回布尔型，true 表示显示该数据，false 表示不显示该数据。向表格中设置和移除过滤器的方法是 addFilter 和 removeFilter 方法。

19.2.7 编辑表格单元

在学习 SWT 的表格时，使用 TableEditor 对象可以设置表格为可编辑状态。JFace 中的

表格也提供了类似编辑器的类。如图 19.6 所示为 JFace 对单元格编辑的类结构示意图。

从图中可以看出，所有的单元格编辑器都是从 `CellEditor` 继承而来的，该类是一个抽象类，创建对象时要使用实现了的子类。下面就以一个程序示例来讲解如何使表格进行编辑。如图 19.7 所示为表格处于编辑状态时的效果图。

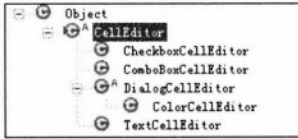


图 19.6 单元格编辑器类继承关系示意图

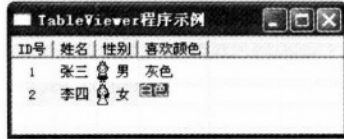


图 19.7 可编辑的表格效果图

实现该效果的代码如下：

```

//设置列属性
table.setColumnProperties( COLUMN_NAME );

//设置单元格编辑器对象数组
CellEditor[] editors = new CellEditor[4];
editors[0] = null;
editors[1] = new TextCellEditor(table.getTable());
editors[2] = new TextCellEditor(table.getTable());
editors[3] = new TextCellEditor(table.getTable());
//设置单元格编辑器
table.setCellEditors(editors);
//设置单元格修改器
table.setCellModifier( new ICellModifier(){

    //如果该列为第一列，设置不能修改
    public boolean canModify(Object element, String property) {
        if ( property.equals(COLUMN_NAME[0]))
            return false;
        return true;
    }
    //当处于编辑状态时所显示的值
    public Object getValue(Object element, String property) {
        PersonEO p = (PersonEO) element;
        if ( property.equals(COLUMN_NAME[1]))
            return p.getName();
        else if ( property.equals(COLUMN_NAME[2]))
            return p.getGender();
        else if ( property.equals(COLUMN_NAME[3]))
            return p.getColor();
        return null;
    }
    //当修改后设置更新数据
    public void modify(Object element, String property, Object value) {
        if (element instanceof Item)
            element = ((Item) element).getData();
    }
}
  
```



```

        PersonEO p = (PersonEO) element;
        if ( property.equals(COLUMN_NAME[1]))
            p.setName( (String)value );
        else if ( property.equals(COLUMN_NAME[2]))
            p.setGender( (String)value );
        else if ( property.equals(COLUMN_NAME[3]))
            p.setColor( (String)value );
        //刷新表格
        table.refresh();
    }
});

```

19.2.8 表格的事件处理

对于 `TableView` 表格，也可以为该表格注册多种事件，常用的事件如下：

1. 鼠标双击事件

通过 `addDoubleClickListener(IDoubleClickListener listener)` 方法为表格注册该事件。例如，以下代码为表格增加了双击事件。

```

table.addDoubleClickListener( new IDoubleClickListener(){
    public void doubleClick(DoubleClickEvent event) {
        StructuredSelection select = (StructuredSelection) event.getSelection();
        PersonEO p = (PersonEO) select.getFirstElement();
        System.out.println(p.getName());
    }
});

```

2. 支持拖动事件

使用以下两个方法可以使表格支持拖动，有关拖动的内容请读者参阅第 10 章。

```

addDragSupport(int operations, Transfer[] transferTypes, DragSourceListener listener)
addDropSupport(int operations, Transfer[] transferTypes, final DropTargetListener listener)

```

3. 其他事件

- ☐ 帮助事件: `addHelpListener(HelpListener listener)`。
- ☐ 选中改变事件: `addSelectionChangedListener(ISelectionChangedListener listener)`。
- ☐ 选中改变后事件: `addPostSelectionChangedListener(ISelectionChangedListener listener)`。
- ☐ 双击打开事件: `addOpenListener(IOpenListener listener)`。

19.2.9 带有复选框表格 (`CheckBoxTableView`)

`JFace` 还有带复选框的表格 (`CheckBoxTableView`)，该表格的每一行都有一个复选框，可以选中或取消选中状态，该表格的效果图如图 19.8 所示。

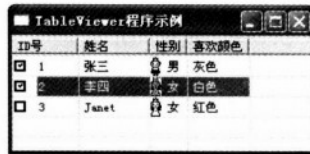


图 19.8 带有复选框的表格

创建 `CheckBoxTableViewer` 表格很简单，因为它是 `TableViewer` 的子类，所以将以上 `TableViewer` 表格程序中创建表格对象的代码改为：

```
CheckboxTableViewer table = CheckboxTableViewer.newCheckList( parent , SWT.FULL_SELECTION);
```

这样就创建了一个带有复选框的表格。另外，`CheckboxTableViewer` 类还提供了对选中状态操作的一些方法，如下：

- ☐ 该行是否已选中：`boolean getChecked(Object element)`。
- ☐ 获得所有已选中的行：`Object[] getCheckedElements()`。
- ☐ 该行是否灰色显示：`getGrayed(Object element)`。
- ☐ 获得所有灰色显示的行：`Object[] getGrayedElements()`。
- ☐ 设置全选和取消全选：`setAllChecked(boolean state)`。
- ☐ 设置全部灰色显示和取消灰色显示：`setAllGrayed(boolean state)`。
- ☐ 设置指定行的选中状态：`setChecked(Object element, boolean state)`。
- ☐ 设置选中多行：`setCheckedElements(Object[] elements)`。
- ☐ 设置指定行是否灰色显示的状态 `setGrayed(Object element, boolean state)`。
- ☐ 设置多行灰色显示：`setGrayedElements(Object[] elements)`。

19.3 树 (TreeViewer)

有了 19.2 节学习 `TableViewer` 的基础，现在再来学习树就容易多了。与表格相比，树比表格有更复杂的数据结构，但基本上创建的步骤与表格类似。下面就以一个产品和类别的树结构为例，学习如何创建复杂的树结构。如图 19.9 所示为该程序运行后的效果图。



图 19.9 产品和分类的树结构示意图

19.3.1 树的基本性质

树是一种基本数据结构，程序中有许多地方都会用到树结构。如操作系统的文件目录就是一个典型的树结构。一个文件系统有一个根目录，根目录下有许多文件夹，每个文件夹又可以包含文件或是文件夹。该文件夹下的文件和文件夹称之为子孙。

本例中使用的是产品和分类，一个类别中可以有子类别，也可以有产品。该类别下的分类和产品都是该类别的子孙。

在实际的项目中，为了平等对待树中的节点对象，通常是抽象出树节点的接口。单就树中的一个节点来说，它有节点的名称、是否有子孙和所有的子孙等性质。这样就需要定义一个树节点的接口，代码如下：

TreeElement.java

```
package com.fengmanfei.ch19;

import java.util.List;
public interface TreeElement{
    public String getName();//节点名称
    public boolean hasChildren();//是否有子孙
    public List getChildren();//获得所有子孙
}
```

存储类别的实体类要实现该接口，类别实体类的具体代码如下：

CategoryEO.java

```
package com.fengmanfei.ch19;

import java.util.ArrayList;
import java.util.List;

public class CategoryEO implements TreeElement{
    private String name ;
    private List lists ;//所有子孙
    public CategoryEO( String name ){
        this.name = name ;
        lists = new ArrayList();
    }
    public String getName() {
        return name;
    }
    public boolean hasChildren() {
        if ( lists.size()>0)
            return true;
        return false;
    }
    public List getChildren() {
```

```
        return lists;
    }
    //添加子孙，可以是类别也可以是产品
    public void add(TreeElement element ) {
        lists.add( element );
    }
}
```

产品实体类也要实现该接口，该类的具体代码如下：

ProductEO.java

```
package com.fengmanfei.ch19;
import java.util.List;

public class ProductEO implements TreeElement{
    private String name ;
    public ProductEO( String name ){
        this.name = name ;
    }
    public String getName() {
        return name;
    }
    public boolean hasChildren() {
        return false;
    }
    public List getChildren() {
        return null;
    }
}
```

有了树的数据实体类，就可以创建树了。

19.3.2 创建树（TreeViewer）

TreeViewer 对象与 19.2 节学习创建 TableView 的步骤类似，大致的步骤是：

```
TreeViewer tree = new TreeViewer(parent);//创建树
tree.setContentProvider(new TreeContentProvider());//设置内容提供者
tree.setLabelProvider(new TreeLabelProvider());//设置标签提供者
tree.setInput(data)//设置初始化的数据
```

下面就来看一下创建树窗口的完整代码，如下：

TreeWindow.java

```
package com.fengmanfei.ch19;

import java.util.ArrayList;
import java.util.List;
```

```
import org.eclipse.jface.action.*;
import org.eclipse.jface.dialogs.InputDialog;
import org.eclipse.jface.viewers.StructuredSelection;
import org.eclipse.jface.viewers.TreeViewer;
import org.eclipse.jface.window.ApplicationWindow;
import org.eclipse.swt.widgets.*;

public class TreeWindow extends ApplicationWindow {
    private TreeViewer tree;//声明树对象
    private Menu popUpMenu;//上下文菜单
    private List data ;//树的初始数据
    public TreeWindow() {
        super(null);
        initData();
    }
    //初始化数据
    private void initData() {
        data = new ArrayList();
        CategoryEO cate1 = new CategoryEO("图书");
        cate1.add( new CategoryEO("小说"));
        cate1.add( new ProductEO("Eclipse"));
        cate1.add( new ProductEO("SWT"));
        data.add( cate1 );
        CategoryEO cate2 = new CategoryEO("音像");
        data.add( cate2 );
        ProductEO product1 = new ProductEO("服装");
        data.add( product1 );
    }
    //设置窗口属性
    protected void configureShell(Shell shell) {
        super.configureShell(shell);
        shell.setSize(200, 300);
        shell.setText("TreeViewer 程序示例");
    }
    //创建控件
    protected Control createContents(Composite parent) {
        initTree(parent);//初始化树
        createContextMenu(parent);//创建上下文菜单
        return parent;
    }
    private void initTree(Composite parent) {
        tree = new TreeViewer(parent);//创建树
        tree.setContentProvider(new TreeContentProvider());//设置树内容提供者
        tree.setLabelProvider(new TreeLabelProvider());//设置标签提供者
        tree.setInput(data);//设置初始化数据
    }
    //创建初始化菜单
    private void createContextMenu(Composite parent) {
```



```

    MenuManager top = new MenuManager();
    top.add( new NewProductAction ());
    top.add( new NewCategoryAction());
    top.add( new Separator());
    top.add( new DeleteAction());
    //创建上下文菜单
    popUpMenu = top.createContextMenu( parent );
    tree.getTree().setMenu(popUpMenu);

}
//自定义方法，获得树当前选中的节点
private TreeElement getSelectElement() {
    StructuredSelection select = (StructuredSelection) tree.getSelection();
    TreeElement element = (TreeElement) select.getFirstElement();
    return element;
}

public static void main(String[] args) {
    TreeWindow test = new TreeWindow();
    test.setBlockOnOpen(true);
    test.open();
    Display.getCurrent().dispose();
}
}

```

最重要的是如何转化树中初始化的数据，对于提供树的内容提供器的类，要实现 `ITreeContentProvider` 接口。这里编写了一个 `TreeContentProvider` 类，作为该树的内容提供者，代码如下：

TreeContentProvider.java

```

package com.fengmanfei.ch19;

import java.util.List;
import org.eclipse.jface.viewers.ITreeContentProvider;
import org.eclipse.jface.viewers.Viewer;

public class TreeContentProvider implements ITreeContentProvider {
    public Object[] getChildren(Object parentElement) {
        return ((TreeElement)parentElement).getChildren().toArray();
    }
    public Object getParent(Object element) {
        return null;
    }
    public boolean hasChildren(Object element) {
        return ((TreeElement)element).hasChildren();
    }
    public Object[] getElements(Object inputElement) {
        return ((List)inputElement).toArray();
    }
}

```

```
public void dispose() { }  
public void inputChanged(Viewer viewer, Object oldInput, Object newInput){}  
}
```

对树的标签提供者，要实现 `ILabelProvider` 接口，本例中编写了 `TreeLabelProvider` 类作为该树的标签提供者，代码如下：

TreeLabelProvider.java

```
package com.fengmanfei.ch19;  
  
import org.eclipse.jface.viewers.ILabelProvider;  
import org.eclipse.jface.viewers.ILabelProviderListener;  
import org.eclipse.swt.graphics.Image;  
import org.eclipse.swt.widgets.Display;  
import com.fengmanfei.util.ImageFactory;  
  
public class TreeLabelProvider implements ILabelProvider{  
  
    public Image getImage(Object element) {  
        if ( element instanceof ProductEO)  
            return ImageFactory.loadImage( Display.getCurrent(), ImageFactory.TOPIC);  
        else if ( element instanceof CategoryEO)  
            return ImageFactory.loadImage( Display.getCurrent(), ImageFactory.TOC_CLOSED);  
        return null;  
    }  
    //显示树节点的名称  
    public String getText(Object element) {  
        return ((TreeElement)element).getName();  
    }  
    public void addListener(ILabelProviderListener listener) {}  
    public void dispose() {  
        ImageFactory.dispose();  
    }  
    public boolean isLabelProperty(Object element, String property) {  
        return false;  
    }  
    public void removeListener(ILabelProviderListener listener) {}  
}
```

19.3.3 对树的操作

最后，再来看一下对树的操作。上下文菜单是通过 `MenuManager` 来创建的，对于每个操作的按钮需要创建一个继承自 `Action` 的类。本例中将介绍这些 `Action` 写作内部类的形式。下面看一下具体的添加和删除数节点的类：

- ❑ 添加产品 `NewProductAction` 类的代码如下：

```
public class NewProductAction extends Action{
    NewProductAction (){
        super("新建产品");
    }
    public void run(){
        TreeElement element = getSelectElement();
        InputDialog input = new InputDialog(getShell(), " 请输入产品", "产品名", "", null);
        int rt = input.open();
        if (rt == OK)
            tree.add(element, new ProductEO(input.getValue()));
    }
}
```

□ 添加类别 NewCategoryAction 类的代码如下：

```
public class NewCategoryAction extends Action{
    NewCategoryAction (){
        super("新建类别");
    }
    public void run(){
        TreeElement element = getSelectElement();
        if (element instanceof ProductEO)
            return;
        InputDialog input = new InputDialog(getShell(), " 请输入类别", "类别名", "", null);
        int rt = input.open();
        if (rt == OK)
            tree.add(element, new CategoryEO(input.getValue()));
    }
}
```

□ 删除节点 DeleteAction 的类代码如下：

```
public class DeleteAction extends Action{
    DeleteAction (){
        super("删除");
    }
    public void run(){
        TreeElement element = getSelectElement();
        tree.remove(element);
    }
}
```

19.4 树和表格的综合示例

学习了树和表格的知识，下面以一个文件浏览器的程序为例，综合树和表格的用法来学习如何进行综合应用。

19.4.1 文件浏览器功能概述

如图 19.10 所示为该文件浏览器程序运行后的效果图。该文件浏览器左侧是一个树结构，右侧是一个表格。该文件浏览器提供的功能有以下方面：

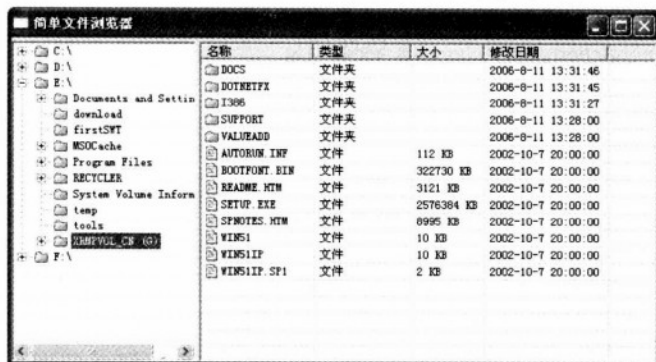


图 19.10 简单文件浏览器

- ❑ 左侧的树中显示的是所有的文件夹，注意不包含文件。
- ❑ 当单击树中某个文件夹后，会在右侧表格中显示出该文件夹中的所有文件。
- ❑ 右侧的表格显示该文件的名称、类型、大小和修改日期。
- ❑ 当单击右侧表格中的文件时，如果是文件，会自动打开文件；若是文件夹，将会打开该文件夹。

19.4.2 程序的整体框架

首先分析一下该文件浏览器的整体框架，整个窗口由一个 SashForm 构成，SashForm 的左侧放置一个 TreeViewer，右侧放置一个 TableViewer。以下是该窗口的主要框架的代码：

FileExplorer.java

```
package com.fengmanfei.ch19;

import java.io.*;
import java.util.*;
import java.util.List;

import org.eclipse.jface.viewers.*;
import org.eclipse.jface.window.ApplicationWindow;
import org.eclipse.swt.SWT;
import org.eclipse.swt.custom.SashForm;
import org.eclipse.swt.graphics.Image;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.program.Program;
```

```

import org.eclipse.swt.widgets.*;
import com.fengmanfei.util.ImageFactory;

public class FileExplorer extends ApplicationWindow {

    private SashForm sash;//分割窗口
    private TreeViewer tree;//树
    private TableViewer table;//表格
    private OpenAction openAction;//事件处理对象

    public FileExplorer() {
        super(null);
        openAction = new OpenAction();
    }
    //初始化窗口内容
    protected Control createContents(Composite parent) {
        this.getShell().setText("简单文件浏览器");
        this.getShell().setMaximized(true);
        sash = new SashForm(parent, SWT.SMOOTH);
        sash.setLayoutData(new GridData(GridData.FILL_BOTH));
        initTree();//初始化树
        initTable();//初始化表格
        sash.setWeights(new int[] { 40, 60 });
        return parent;
    }
    //获得当前树选中的行
    public Object getTreeSelection(){
        IStructuredSelection selection = (IStructuredSelection) tree.getSelection();
        if (selection.size() != 1)
            return null;
        return selection.getFirstElement();
    }
    //获得当前表格选中的行
    public Object getTableSelection(){
        IStructuredSelection selection = (IStructuredSelection) table.getSelection();
        if (selection.size() != 1)
            return null;
        return selection.getFirstElement();
    }
    public static void main(String[] args) {
        FileExplorer test = new FileExplorer();
        test.setBlockOnOpen(true);
        test.open();
        Display.getCurrent().dispose();
    }
}

```

该程序中，将初始化树写成了一个方法 `initTree`，将初始化表格也写成了一个方法 `initTable`。这两个方法将在下文讲述。另外，将获得树中当前选中节点的操作也写成了一个

方法 `getTreeSelection`，对表格当前选中的行也写成了一个方法 `getTableSelection`。

19.4.3 初始化树

下面就来看一下初始化树 `initTree` 的具体实现代码。如下：

```
//初始化树
private void initTree() {
    tree = new TreeViewer(sash);
    //设置树内容提供者
    tree.setContentProvider(new FileTreeContentProvider());
    //设置树标签提供者
    tree.setLabelProvider(new FileTreeLabelProvider());
    //设置树初始化的数据，这里没有任何意义
    //使用的是根目录的数据在内容提供者中
    tree.setInput("root");
    //注册树选中事件
    tree.addSelectionChangeListener(openAction);
};
```

关键是要看如何提供树中的数据和显示的方式，本例中创建的类均为内部类实现。首先看一下树的内容提供者 `FileTreeContentProvider` 类的实现：

```
public class FileTreeContentProvider implements ITreeContentProvider {
    public Object[] getChildren(Object element) {
        //过滤后只显示文件夹
        return ((File) element).listFiles(new AllowOnlyFoldersFilter());
    }
    public Object[] getElements(Object element) {
        File[] roots = File.listRoots();//获得根目录的文件
        List rootFolders = new ArrayList();
        //过滤根目录，只显示根目录中的文件夹
        for (int i = 0; i < roots.length; i++) {
            if (roots[i].isDirectory())
                rootFolders.add(roots[i]);
        }
        return rootFolders.toArray();
    }
    public boolean hasChildren(Object element) {
        Object[] obj = getChildren(element);
        return obj == null ? false : obj.length > 0;
    }
    public Object getParent(Object element) {
        return ((File) element).getParentFile();
    }
    public void dispose() {
    }
    public void inputChanged(Viewer viewer, Object oldInput, Object newInput) {}
}
```

这里需要注意的是，对初始化数据的处理，也就是 `getElements` 方法中的代码。通过 `File` 类的静态方法 `File.listFiles()` 可以获得本机根目录，但根目录可能不全是文件夹，这时就需要过滤一下判断是否是文件夹，然后返回所有根目录的文件夹。

另外，在树中只能显示文件夹，而不显示文件，所以在 `getChildren` 方法中，返回的是 `listFiles(new AllowOnlyFoldersFilter())`，其中 `AllowOnlyFoldersFilter` 是一个过滤器对象，具体使用方法请参阅 JDK API 文档，这里给出该类的实现代码，如下：

```
public class AllowOnlyFoldersFilter implements FileFilter {
    //如果是文件夹，则显示
    public boolean accept(File pathname) {
        return pathname.isDirectory();
    }
}
```

然后再看一下树的标签提供者 `FileTreeLabelProvider` 类实现的代码：

```
public class FileTreeLabelProvider implements ILabelProvider {
    public Image getImage(Object element) {
        File file = (File) element;
        if (file.isDirectory())
            return ImageFactory.loadImage(Display.getCurrent(), ImageFactory.FOLDER);
        return ImageFactory.loadImage(Display.getCurrent(), ImageFactory.FILE);
    }
    public String getText(Object element) {
        //对于根目录中的文件夹，没有名称，此时要显示路径
        String text = ((File) element).getName();
        if (text.length() == 0) {
            text = ((File) element).getPath();
        }
        return text;
    }
    public void addListener(ILabelProviderListener listener) { }
    public void dispose() {
        ImageFactory.dispose();
    }
    public boolean isLabelProperty(Object element, String property) {
        return false;
    }
    public void removeListener(ILabelProviderListener listener) {}
}
```

这里需要注意的是，在 `getText` 方法中，对于根目录中的文件是没有文件名的，这时要用路径来显示。

19.4.4 初始化表格

初始化表格 `initTable` 的具体实现代码如下：

//初始化表格

```
private void initTable() {
    table = new TableViewer(sash);
    //创建表头
    new TableColumn(table.getTable(), SWT.LEFT).setText("名称");
    new TableColumn(table.getTable(), SWT.LEFT).setText("类型");
    new TableColumn(table.getTable(), SWT.LEFT).setText("大小");
    new TableColumn(table.getTable(), SWT.LEFT).setText("修改日期");
    for (int i = 0; i < table.getTable().getColumnCount(); i++) {
        table.getTable().getColumn(i).pack();
    }
    //设置表头和网格线可见
    table.getTable().setHeaderVisible(true);
    table.getTable().setLinesVisible(true);
    //设置表格内容提供者
    table.setContentProvider(new FileTableContentProvider());
    //设置表格标签提供者
    table.setLabelProvider(new FileTableLabelProvider());
    //设置排序对象
    table.setSorter(new FileSorter());
    //注册双击事件
    table.addClickListener(openAction);
}
```

表内容提供者 FileTableContentProvider 的代码如下:

```
class FileTableContentProvider implements IStructuredContentProvider {
    public void dispose() {}
    public void inputChanged(Viewer viewer, Object oldInput, Object newInput) {}
    public Object[] getElements(Object inputElement) {
        File file = (File) inputElement;
        return file.listFiles();
    }
}
```

表的标签提供者 FileTableLabelProvider 的代码如下:

```
class FileTableLabelProvider implements ITableLabelProvider {
    public Image getColumnImage(Object element, int columnIndex) {
        File file = (File) element;
        if (columnIndex == 0) {
            if (file.isDirectory())
                return ImageFactory.loadImage(Display.getCurrent(), ImageFactory.FOLDER);
            else
                return ImageFactory.loadImage(Display.getCurrent(), ImageFactory.FILE);
        }
        return null;
    }
    public String getColumnText(Object element, int columnIndex) {
        File file = (File) element;
        if (columnIndex == 0 )
```

```

        return file.getName();//显示文件名称
    else if (columnIndex == 1) {
        //显示文件类型
        if (file.isDirectory())
            return "文件夹";
        else
            return "文件";
    } else if (columnIndex == 2) {
        //如果是文件夹，则没有大小
        if (file.isDirectory())
            return "";
        else
            return file.length() + " KB";
    } else if (columnIndex == 3) {
        //显示日期
        Date date = new Date(file.lastModified());
        return date.toLocaleString();
    }
    return null;
}
public void addListener(ILabelProviderListener listener) {}
public void dispose() {
    ImageFactory.dispose();
}
public boolean isLabelProperty(Object element, String property) {
    return false;
}
public void removeListener(ILabelProviderListener listener) {}
}

```

这里需要注意的是，对于文件夹是没有大小的，需要作一下判断。另外，通过 `file.lastModified()` 获得文件修改的时间要使用本地的时间格式显示。

另外，为了使表格中首先显示文件夹然后再显示文件，还要为表格设置排序对象 `FileSorter`。该类的代码如下：

```

public class FileSorter extends ViewerSorter {
    public int category(Object element) {
        return ((File) element).isDirectory() ? 0 : 1;
    }
}

```

19.4.5 程序的事件处理

最后看一下树和表格事件的处理。本程序中将表格和树的事件集中处理，编写了一个 `OpenAction` 类，可以在初始化树和初始化表格的方法中找到以下两行代码：

```

table.addDoubleClickListener(openAction);
tree.addSelectionChangedListener(openAction);

```


这里都是使用的 `OpenAction` 对象，以下是该类实现了 `ISelectionChangedListener` 和 `IDoubleClickListener` 接口，具体的代码如下：

```
public class OpenAction implements ISelectionChangedListener, IDoubleClickListener {
    //选中树中一个节点时，所处理的事件
    public void selectionChanged(SelectionChangedEvent event) {
        table.setInput(getTreeSelection());
    }
    //双击表格中一列时
    public void doubleClick(DoubleClickEvent event) {
        Object selection = getTableSelection();
        if (selection==null)
            return;
        File file = (File) selection;
        if (file.isFile()) {
            //如果是文件，打开该文件
            Program.launch(file.getAbsolutePath());
        } else if (file.isDirectory()) {
            //如果是文件夹，在表格中打开文件夹
            table.setInput( selection );
        }
    }
}
```

在执行打开文件的操作时，使用第 13 章中介绍的打开应用程序的方法。

19.5 列表 ListView

理解了表格和树这样复杂的控件，再来学习列表 `ListView` 就容易多了。这里只给出创建 `ListView` 简单的代码，就不使用程序来讲解了。创建一个列表的代码如下：

```
ListView list = new ListView(parent);
list.setContentProvider(MyContentProvider());
list.setLabelProvider(MyLabelProvider());
list.setInput();//设置初始化的数据
```

其中，`MyContentProvider` 要实现 `IStructuredContentProvider` 接口，`MyLabelProvider` 要实现 `ILabelProvider` 接口。但是需要注意的是，`ListView` 底层是使用 `List` 列表来实现的，`List` 中不能同时显示图标和文字。

另外，以下列举了该类中其他常用的一些方法：

- ☐ 获得封装的 `List` 对象：`getList()`。
- ☐ 在指定索引插入一行：`listAdd(String string, int index)`。
- ☐ 取消全选：`listDeselectAll()`。
- ☐ 获得选项的总数目：`listGetItemCount()`。

- ❑ 获得选中选项的索引值: `listGetSelectionIndices()`。
- ❑ 删除指定索引的选项: `listRemove(int index)`。
- ❑ 删除所有选项: `listRemoveAll()`。
- ❑ 修改指定索引的选项值: `listSetItem(int index, String string)`。
- ❑ 选中指定的索引值选项: `listSetSelection(int[] ixes)`。
- ❑ 显示已选中的选项: `listShowSelection()`。

19.6 本章小结

本章学习了 MVC 模式的表格 (TableViewer)、树 (TreeViewer) 和列表 (ListViewer)，这些都是在实际进行项目开发时最常用的控件。但请读者牢记，这些类最底层封装的是 Table、Tree 和 List 控件。当这些 Viewer 类的功能不能满足需求时，也可以获得这些最底层控件的对象，来实现这些功能。



第 20 章 JFace 的工具类

本章将详细讲述 JFace 中的一些工具类,包括改善的资源管理机制、对线程的创建和常用的一些工具方法。这些是 JFace 提供的一些常用的工具类,了解这些类后可以在以后的项目中直接使用。

20.1 JFace 资源管理机制

使用 SWT 的程序,资源的释放问题始终是一个很头疼也很重要的问题。幸运的是 JFace 中已经提供了很好的解决方案,解决资源的管理问题。使用 JFace 的资源管理类的好处是我们只需要使用这些资源,而不用去管释放的问题。当 Display 对象释放时,自动将释放这些资源。这与 Java 的垃圾回收机制类似,不用管何时销毁这个资源,当释放 Display 对象时将自动地释放资源。JFace 的资源管理的类都在 org.eclipse.jface.resource 包中。

20.1.1 图像描述符 (ImageDescriptor)

JFace 中的 ImageDescriptor 类提供了一个轻量级存储图像信息的对象。使用 ImageDescriptor 类的好处是,当所构造图像资源不存在时,不会抛出异常,而是将图片以一个红色方框代替。

ImageDescriptor 是一个抽象类,不能够直接创建对象,但可以通过封装的一些静态方法来创建。这些创建 ImageDescriptor 对象的静态方法有以下几种:

1. ImageDescriptor createFromFile(Class location, String filename)

其中 location 为类对象, filename 为图像文件名,此种方法适用于从磁盘保存的文件中获得图片资源。使用此方法的代码如下:

```
ImageDescriptor image = ImageDescriptor.createFromFile( ImageWindow.class , "eclipse.jpg" );
```

这里, eclipse.jpg 图片要位于 ImageWindow.java 统一目录下,才可以正常显示图片。如图 20.1 所示,显示了 ImageWindow.java 文件与图像文件的路径位置。

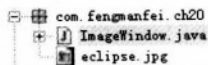


图 20.1 类文件与图像文件的路径结构

2. ImageDescriptor createFromURL(URL url)

其中 url 为 Web 中的图片的地址路径。此种方法适用于从网页中获得一个图片资源文

件。使用此方法的代码如下：

```
ImageDescriptor urlImage = null;
try {
    URL url = new URL("http://www.eclipse.org/eclipse.org-common/themes/Phoenix/images/
header_logo.gif");
    urlImage = ImageDescriptor.createFromURL(url);
} catch (MalformedURLException e) {
    e.printStackTrace();
}
```

其中 url 对象可以指定网页中的一个图片地址来获得该图片。有关 URL 类的使用方法请读者参阅 JDK 文档。

3. ImageDescriptor createFromImage(Image img)

其中 img 为已创建的图像对象，此方法适用于已经创建了图像对象的图像资源。使用此方法的代码如下：

```
Image i = new Image(Display.getCurrent(), "F:\\icon\\edit.gif");
ImageDescriptor id = ImageDescriptor.createFromImage(i);
```

但是，需要注意的是，在使用 Image 对象构造 ImageDescriptor 之前要确保该 Image 没有被释放掉，否则会抛出异常。例如，以下代码运行后会抛出异常错误。

```
Image i = new Image(Display.getCurrent(), "F:\\icon\\edit.gif");
i.dispose();
ImageDescriptor id = ImageDescriptor.createFromImage(i);
```

4. ImageDescriptor createFromImage(Image img, Device theDevice)

其中 theDevice 为指定的图像所在设备，该方法适用于不是 Display 对象的资源。如果不指定 theDevice 对象，默认使用的是 Display 对象。使用此方法的代码如下：

```
Image i = new Image(Display.getCurrent(), "F:\\icon\\edit.gif");
Printer printer = new Printer();
ImageDescriptor id = ImageDescriptor.createFromImage(i, printer);
printer.dispose();
```

5. ImageDescriptor createFromImageData(ImageData data)

其中 data 为 ImageData 对象，该方法适用于由 ImageData 对象构造的 ImageDescriptor 对象。使用此方法的代码如下：

```
ImageData data = new ImageData("F:\\icon\\edit.gif");
ImageDescriptor imageData = ImageDescriptor.createFromImageData( data );
```

有关 ImageData 对象的创建和使用的方法请读者参阅第 12 章的有关内容。

6. ImageDescriptor createWithFlags(ImageDescriptor originalImage, int swtFlags)

其中 swtFlags 的值可以是 SWT . IMAGE_COPY、SWT . IMAGE_DISABLE 或者 SWT . IMAGE_GRAY。originalImage 是原始的 ImageDescriptor 对象。使用此方法的代码如下：

```
ImageDescriptor image = ImageDescriptor.createFromFile( ImageWindow.class , "eclipse.jpg" );  
ImageDescriptor newImage = ImageDescriptor.createWithFlags(image,SWT.IMAGE_DISABLE)
```

创建了图像描述符后，怎样通过该描述符对象获得相应的 Image 对象呢？通过以下所示方法便可以获得图像资源的对象。

7. Image createImage()

通过这种方法获得图片对象，如果该图片资源不存在，将以一个红色方框表示该图片。例如，以下代码为一个按钮设置图片。

```
ImageDescriptor image = ImageDescriptor.createFromFile( ImageWindow.class , "eclipse.jpg" );  
Button bt = new Button( parent , SWT.NONE);  
bt.setImage( image.createImage());
```

20.1.2 图像注册器 (ImageRegistry)

在 9.3.1 节中，为了实现对图像资源的简单管理，专门写过一个 ImageFactory 类来管理图片，大致的原理是通过一个 Hashtable 对象来集中管理图片，但自己编写的这个类虽然不能自动地释放图片资源，但也实现了简单管理机制。

JFace 中提供的 ImageRegistry 类改进了这些缺点，它底层是由一个 Map 对象管理，通过 put 方法可以将一个图片对象放入到 Map 中，通过方法可以将 Map 中存放的图片对象取出来，通过 remove 方法可以移除 Map 中存放的图片对象。

使用 ImageRegistry 类来管理图片的最大好处是，不需要显式地释放存放 ImageRegistry 对象中的图片资源，当 Display 对象释放时，会自动释放 ImageRegistry 对象中存放的图片资源，因为这些都交给了底层的 JFace 资源管理机制了。

下面以一个程序来说明如何使用 ImageRegistry 类，该程序显示 3 个图片标签，程序运行后的效果如图 20.2 所示。

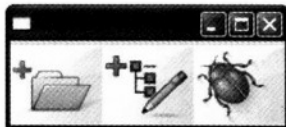


图 20.2 使用 ImageRegistry 对象程序运行后效果图

该程序的具体代码如下：

ImageRegistryTest.java

```
package com.fengmanfei.ch20;  
  
import org.eclipse.jface.resource.*;  
import org.eclipse.jface.window.ApplicationWindow;  
import org.eclipse.swt.SWT;  
import org.eclipse.swt.layout.FillLayout;  
import org.eclipse.swt.widgets.*;
```

```
public class ImageRegistryTest extends ApplicationWindow {

    private static final String ONE = "one";
    private static final String TWO = "two";
    private static final String THREE = "three";
    public ImageRegistryTest() {
        super(null);
    }

    protected Control createContents(Composite parent) {
        Composite composite = new Composite(parent, SWT.NONE);
        composite.setLayout(new FillLayout());

        //将 3 个文件放入到 ImageRegistry 中
        ImageRegistry ir = new ImageRegistry();
        ir.put(ONE, ImageDescriptor.createFromFile(ImageRegistryTest.class, "one.gif"));
        ir.put(TWO, ImageDescriptor.createFromFile(ImageRegistryTest.class, "two.gif"));
        ir.put(THREE, ImageDescriptor.createFromFile(ImageRegistryTest.class, "three.gif"));

        //取出图片
        Label label = new Label(composite, SWT.NONE);
        label.setImage(ir.get(ONE));
        label = new Label(composite, SWT.NONE);
        label.setImage(ir.get(TWO));
        label = new Label(composite, SWT.NONE);
        label.setImage(ir.get(THREE));

        getShell().pack();
        return composite;
    }

    public static void main(String[] args) {
        ImageRegistryTest test = new ImageRegistryTest();
        test.setBlockOnOpen(true);
        test.open();
        Display.getCurrent().dispose();
    }
}
```

由以上程序可以总结出使用 ImageRegistry 对象时常见的问题，有以下几点：

- ❑ 在使用 ImageRegistry 对象管理图片的过程中并没有发现 dispose 这样的代码，这是使用 ImageRegistry 对象的最大优势。
- ❑ 对于放入到 ImageRegistry 对象中的 key，一般使用字符常量，这是因为在取出图片时使用常量容易获得该图片。
- ❑ 在实际的项目中一般不会在一个类中创建一个 ImageRegistry 对象，更多的做法是将 ImageRegistry 对象集中在一个类。这样其他类中的对象也可以共享该

ImageRegistry 对象的图片资源。有关这部分的设计，第 21 章将会涉及这些内容。

20.1.3 字体描述符和字体注册器

字体与图像一样也属于系统的资源，并且对应的有字体描述符 (FontDescriptor) 和字体注册器 (FontRegistry)。

1. 字体描述符 (FontDescriptor)

字体描述符 (FontDescriptor) 与 ImageDescriptor 的功能和用法类似，这里只列举出所使用的静态方法，就不多详细介绍了。

- ❑ FontDescriptor createFrom(Font font): 通过 Font 对象构造。
- ❑ FontDescriptor createFrom(FontData[] data): 通过 FontData 数组对象构造。
- ❑ FontDescriptor createFrom(FontData data): 通过 FontData 对象构造。
- ❑ FontDescriptor createFrom(String name, int height, int style): 其中 name 为字体的名称、height 为字体的大小、style 的值可以有 SWT.NORMAL 表示正常字体，也可以指定加粗 SWT.BOLD 或倾斜 SWT.ITALIC。

创建完字体描述符对象后，通过以下方法可以创建字体对象：

```
Font createFont(Device device)
```

注意，该方法需要捕获异常。以下是为按钮设置字体的具体代码：

```
FontData fontData = new FontData("隶书",30, SWT.BOLD);
FontDescriptor fd = FontDescriptor.createFrom(fontData);
Button bt = new Button( parent , SWT.NONE);
bt.setText("设置字体");
try {
    bt.setFont( fd.createFont(Display.getCurrent()));
} catch (DeviceResourceException e) {
    e.printStackTrace();
}
```

程序运行后，按钮中显示的字体的效果如图 20.3 所示。

2. 字体注册器 (FontRegistry)

字体注册器 (FontRegistry) 与图像注册器 (ImageRegistry) 类似，但与之不同的是使用的 put 方法放入的是 FontData 数组对象，该方法的具体参数是：

```
put(String symbolicName, FontData[] fontData)
```

另外在取出字体时可以分别取出不同类型的字体，方法如下：

- ❑ get(String symbolicName): 取出字体，若没有该 key 指定的字体时，返回默认的字体。
- ❑ getBold(String symbolicName): 取出加粗的字体。
- ❑ getItalic(String symbolicName): 取出倾斜的字体。



图 20.3 设置字体后的按钮文字

另外可以使用 `defaultFont()` 方法获得默认的字体系对象, 使用 `hasValueFor(String fontKey)` 方法判断该字体注册器中是否有该 `key` 对应的字体系对象。

20.1.4 颜色描述符和颜色注册器

颜色描述符和颜色注册器对应的类分别为 `ColorDescriptor` 和 `ColorRegistry` 类。这里只将这两个类的方法作一下说明, 用法与字体描述符和字体注册器的用法相同。

1. 颜色描述符 (`ColorDescriptor`)

- ❑ `ColorDescriptor createFrom(Color toCreate)`: 由 `Color` 对象构造。
- ❑ `ColorDescriptor createFrom(RGB toCreate)`: 由 `RGB` 对象构造。
- ❑ `ColorDescriptor createFrom(Color toCreate, Device originalDevice)`: 由 `Color` 对象和指定的 `Device` 对象构造。

另外, 通过 `ColorDescriptor` 对象获得一个颜色的方法是 `createColor(Device device)`。

2. 颜色注册器 (`ColorRegistry`)

- ❑ `put(String symbolicName, RGB colorData)`: 注册一个 `RGB` 颜色。
- ❑ `Color get(String symbolicName)`: 获得该 `key` 的颜色对象。
- ❑ `RGB getRGB(String symbolicName)`: 获得该 `key` 的 `RGB` 对象。
- ❑ `ColorDescriptor getColorDescriptor(String symbolicName)`: 获得该 `key` 的 `ColorDescriptor` 对象。
- ❑ `boolean hasValueFor(String colorKey)`: 判断是否注册了该 `key`。

下面以一个具体的程序来说明如何使用字体注册器和颜色注册器。程序运行后分别显示字体和背景色不同的两个标签, 程序运行后的效果如图 20.4 所示。



图 20.4 不同样式的字体和背景色

该程序实现的代码如下:

FontAndColorTest.java

```
package com.fengmanfei.ch20;

import org.eclipse.jface.resource.*;
import org.eclipse.jface.window.ApplicationWindow;
import org.eclipse.swt.SWT;
import org.eclipse.swt.graphics.*;
import org.eclipse.swt.layout.FillLayout;
import org.eclipse.swt.widgets.*;

public class FontAndColorTest extends ApplicationWindow {
```

```
//使用的 key 常量
private static final String FONT_ONE = "font_one";
private static final String FONT_TWO = "font_two";
private static final String COLOR_BLUE = "blue";
private static final String COLOR_GREEN = "green";
//FontRegistry 对象
private static FontRegistry fontRegistry;
//ColorRegistry 对象
private static ColorRegistry colorRegistry;
public FontAndColorTest() {
    super(null);
}

private void loadFontAndColor() {
    //初始化字体和颜色注册器对象
    fontRegistry = new FontRegistry();
    colorRegistry = new ColorRegistry();
    //添加字体
    FontData fontData = new FontData("隶书",24, SWT.BOLD);
    fontRegistry.put(FONT_ONE,new FontData[]{ fontData });
    fontData = new FontData("楷体",30, SWT.NORMAL);
    fontRegistry.put(FONT_TWO,new FontData[]{ fontData });
    //添加颜色
    colorRegistry.put(COLOR_BLUE, new RGB(0,128,255));
    colorRegistry.put(COLOR_GREEN, new RGB(128,255,128));
}

protected Control createContents(Composite parent) {
    //初始化字体和颜色
    loadFontAndColor();
    Composite composite = new Composite(parent, SWT.NONE);
    composite.setLayout(new FillLayout());
    //设置两个字体和颜色不同的标签
    Label label = new Label( composite , SWT.NONE);
    label.setText("样式一");
    label.setFont( fontRegistry.getItalic(FONT_ONE));
    label.setBackground( colorRegistry.get(COLOR_BLUE ));

    label = new Label( composite , SWT.NONE);
    label.setText("样式二");
    label.setFont( fontRegistry.getBold(FONT_TWO));
    label.setBackground( colorRegistry.get(COLOR_GREEN ));
    return parent;
}

public static void main(String[] args) {
    FontAndColorTest test = new FontAndColorTest();
}
```

```
        test.setBlockOnOpen(true);
        test.open();
        Display.getCurrent().dispose();
    }
}
```

20.1.5 JFace 的资源管理器 (JFaceResources)

如果读者查看 JFace 类的源代码经常可以使用 JFaceResources 类来获取类中使用的资源。JFaceResources 类是 JFace 内部使用的一个管理所有的字体、图像、颜色和文本资源的类。该类的设计模式采用了 Singleton 模式，简单地说对资源的管理采用的是全局变量，该类只有一个实例对象。在实际的项目中，可以利用 JFaceResources 提供的一些管理机制进行管理。

1. 有关字体

可以通过 getFontRegistry() 获得 FontRegistry 对象的引用，例如以下代码：

```
FontRegistry fontRegistry = JFaceResources.getFontRegistry();
```

另外 JFaceResources 还提供了获得系统字体的一些方法，如下所示：

```
//获得 Banner 字体
Font bannerFont = JFaceResources.getBannerFont();
//获得默认字体
Font defaultFont = JFaceResources.getDefaultFont();
//获得对话框字体
Font dialogFont = JFaceResources.getDialogFont();
//获得标题的字体
Font headerFont = JFaceResources.getHeaderFont();
//获得文本的字体
Font textFont = JFaceResources.getTextFont();
```

2. 有关颜色

可以通过 getFontRegistry() 获得 FontRegistry 对象的引用，例如以下代码：

```
ColorRegistry colorRegistry = JFaceResources.getColorRegistry();
```

3. 有关图像

可以通过 getFontRegistry() 获得 FontRegistry 对象的引用，例如以下代码：

```
ImageRegistry imageRegistry = JFaceResources.getImageRegistry();
```

4. 有关文本

可以使用 getString 方法获得 ResourceBundle 中的本地字符，一般在对话框中的按钮中经常用到这些已内置的字符，如以下代码所示为获得一个“完成”的本地字符串：

```
JFaceResources.getString(IDialogConstants.FINISH_LABEL);
```

20.1.6 字符转换工具类 (StringConverter)

StringConverter 提供了一些将基本类型数值转化成字符串 String 类型的方法和将字符串转化成其他类型的方法。例如, 像 RGB 这样的存储颜色的对象, 可能会保存在首选项设置的文件中, 这时就需要转化成字符串的形式。当取出这些保存颜色的字符时又需要转换成 RGB 对象, 这时就可以直接使用 StringConverter 类进行转化。例如, 以下代码就是将 RGB 对象转化成字符, 再将字符转化成 RGB 对象的过程。

```
//转化成字符
String s = StringConverter.asString(new RGB(128,128,128));
System.out.println(s);
//字符转化成 RGB 对象
RGB rgb = StringConverter.asRGB(s);
```

该类的方法中都是以 asXXX 开头的, 其中 XXX 为转化后的对象的类型。该类可以转化为 String 类型的有 boolean、int、long、float、double、org.eclipse.swt.graphics.Point、org.eclipse.swt.graphics.Rectangle、org.eclipse.swt.graphics.RGB 和 org.eclipse.swt.graphics.FontData。

20.2 类型检查的工具类

JFace 内部使用的一些工具类都打包在 org.eclipse.jface.util 中, 了解了这些工具类, 在编写程序的过程中可以直接使用。Assert 工具类中提供了一些类型检查的方法, 例如以下方法可以判断一个对象是否为 null 值, 如果为 null, 则抛出所对应的异常信息。

```
isNotNull(Object object, String message)
```

其中 object 为判断的对象, message 为当为 null 值时抛出的错误消息。例如以下代码:

```
String s = null;
Assert.isNotNull(s, "错误消息");
```

除了判断是否为 null 值, 还有以下常用的判断类型的方法:

- ❑ boolean isLegal(boolean expression): 表达式是否有效。
- ❑ isLegal(boolean expression, String message): 表达式是否有效, 若无效则抛出指定的错误消息。
- ❑ isNotNull(Object object): 判断是否为 null 值, 不抛出错误提示消息。
- ❑ boolean isTrue(boolean expression): 是否返回 true。
- ❑ isTrue(boolean expression, String message): 是否返回 true, 如果不返回 true 则抛出错误消息。

20.3 本章小结

本章介绍了 JFace 内部使用的一些工具类, 了解了这些类的用法可以在开发自己的项目时直接使用。当然学习这些类的用法不是目的, 关键要能够借鉴它设计的一些模式, 比如资源管理上, JFaceResources 就使用了 Singleton 的设计模式, 这些模式可以在开发自己的项目中使用。



第 21 章 文本处理

JFace 提供的文本处理功能非常强大，体现在 Eclipse 平台中编辑器的部分。读者在使用 Eclipse 进行开发时会深刻地体会到 Eclipse 编辑器强大的功能，例如代码的折叠、内容提示、代码的格式化等，这些都是基于 JFace 的文本处理框架的。由于关于文本处理的知识非常多，所以在本章中笔者将以一个完整的项目案例——JavaScript 的编辑器来带领读者快速地认识 JFace 中有关文本处理的部分。

21.1 文本处理概述

JFace 中有关文本处理的部分分别打包在以下所示的两个包中，在进行开发之前请确保这两个包已导入到构建路径中。

- ❑ org.eclipse.jface.text_3.1.2.jar。
- ❑ org.eclipse.text_3.1.1.jar。

文本的处理在各种语言的编辑器中经常使用，功能主要有语法着色、语法检查、内容辅助提示等，JFace 的文本处理框架的功能远不止这些，功能非常强大。

由于文本处理的这些内容非常多，如果要详细介绍，都可以写成一本书了。所以这里为了让读者更快地理解这部分内容，笔者将结合项目来介绍。

21.2 项目实战：JavaScript 编辑器

JavaScript 编辑器是 JavaScript 网页中使用的脚本，该编辑器将实现对 JavaScript 脚本语法着色、语法提示等功能。

21.2.1 主窗口预览

首先来看一下程序运行后的效果图，如图 21.1 所示为程序运行后的主窗口效果图。

该编辑器实现的主要功能有以下方面：

- ❑ 文件的打开、保存和打印。
- ❑ 可自定义不同关键字的颜色，如注释的颜色、字符串的颜色、JavaScript 关键字的颜色和一些 JavaScript 内置对象的颜色。
- ❑ 可自定义显示文本的字体格式。

- ❑ 对文本进行撤销和重复操作。
- ❑ 可对文本进行查找和替换功能。
- ❑ 当输入代码的过程中，可以在提示 JavaScript 语法中内置对象的内容提示功能。



图 21.1 主窗口程序效果图

21.2.2 项目文件结构

下面对整个程序中所涉及的文件类进行简要介绍，表 21.1 所示为该程序中涉及的类以及所表示的意义。

表 21.1 类结构表

包名: com.fengmanfei.ch21s		
包含的类	Constants.java	保存了程序中使用的一些字符串常量
	EventManager.java	集中处理事件的类
	JSCodeScanner.java	搜索代码规则的类
	JSEditor.java	程序的主窗口类
	JSEditorConfiguration.java	编辑器配置类
	JSEditorTestDrive.java	程序的入口类
	JSPreferencePage.java	设置首选项页面
	ObjectContentAssista.java nt	内容助手类
	ObjectDetector.java	搜索内置对象字符类
	KeywordDetector.java	搜索关键字对象字符类
	PersistentDocument.java	持久化文档类
	ResourceManager.java	程序中涉及的一些资源的管理
com.fengmanfei.ch21.action		
包含的类	HelpAction.java	帮助菜单项
	OpenAction.java	打开文件菜单项
	PreferenceAction.java	打开首选项对话框项

续表

com.fengmanfei.ch21.action		
包含的类	PrintAction.java	打印菜单项
	RedoAction.java	撤销菜单项
	SaveAction.java	保存菜单项
	SearchAction.java	查找替换菜单项
	UndoAction.java	重复此单项类
com.fengmanfei.ch21.dialog		
包含的类	AboutDialog.java	关于对话框
	FindAndReplace.java	查找替换对话框

21.3 主窗口模块

主窗口模块是本程序的主界面，它继承自 `ApplicationWindow`，是一个应用程序窗口，此外窗口中还添加了菜单栏和工具栏。

21.3.1 代码实现

该主窗口程序中显示代码的控件是 `SourceViewer` 对象。该程序中实现的具体代码如下：

JSEditor.java

```
package com.fengmanfei.ch21;

import org.eclipse.jface.action.*;
import org.eclipse.jface.preference.*;
import org.eclipse.jface.text.*;
import org.eclipse.jface.text.source.*;
import org.eclipse.jface.util.*;
import org.eclipse.jface.window.*;
import org.eclipse.swt.SWT;
import org.eclipse.swt.graphics.*;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.widgets.*;
import com.fengmanfei.ch21.action.*;

public class JSEditor extends ApplicationWindow implements IPropertyChangeListener {

    private PersistentDocument document;//数据文档对象
    private SourceViewer viewer;//显示文本控件对象
    private EventManager eventManager;//事件管理器
    private PreferenceStore preference;//保存首选项设置的对象
    private IUndoManager undoManager;//撤销与重复管理器
    private JSEditorConfiguration configuration;//文本控件的配置对象
```

```

public JSEditor() {
    super(null);
    eventManager = new EventManager(this); // 初始化事件管理器
    this.addMenuBar(); // 添加菜单
    this.addToolBar( SWT.FLAT ); // 添加工具栏
}

protected void configureShell(Shell shell) {
    super.configureShell(shell);
    shell.setText("JavaScript 编辑器");
    shell.setSize(600, 400);
}

protected Control createContents(Composite parent) {
    Composite top = new Composite(parent, SWT.NONE);
    top.setLayout( new FillLayout());
    // 初始化文档对象
    document = new PersistentDocument();
    // 初始化源文件显示控件对象
    viewer = new SourceViewer(top, new VerticalRuler(10), SWT.V_SCROLL | SWT.H_SCROLL);
    // 初始化文档的配置对象
    configuration = new JSEditorConfiguration(this);
    viewer.configure(configuration); // 设置文档配置
    viewer.setDocument(document); // 设置文档
    undoManager = new DefaultUndoManager(100); // 初始化撤销管理器对象，默认可撤销 100 次
    undoManager.connect(viewer); // 将该撤销管理器应用于文档
    // 初始化代码中的字体
    initCodeFont();
    return parent;
}

// 初始化代码的字体
private void initCodeFont() {
    // 定义一个默认的字
    FontData defaultFont = new FontData("Courier New", 10, SWT.NORMAL);
    // 如果从首选项读出的字体有异常，则使用默认的字
    FontData data = StringConverter.asFontData(ResourceManager.getPreferenceStore().
getString(Constants.CODE_FONT), defaultFont);
    // 创建字体
    Font font = new Font( this.getShell().getDisplay(), data);
    viewer.getTextWidget().setFont( font ); // 设置字体
}

// 初始化菜单项
protected MenuManager createMenuManager() {
    MenuManager top = new MenuManager();
    MenuManager fileMenu = new MenuManager("文件(&F)");
    MenuManager editMenu = new MenuManager("编辑(&E)");
    MenuManager helpMenu = new MenuManager("帮助(&H)");

    fileMenu.add( new OpenAction(this) );
    fileMenu.add( new SaveAction(this) );
}

```



```
fileMenu.add( new Separator());
fileMenu.add( new PrintAction(this) );

editMenu.add( new UndoAction(this));
editMenu.add( new RedoAction(this));
editMenu.add( new Separator());
editMenu.add( new SearchAction(this));
editMenu.add( new Separator());
editMenu.add( new PreferenceAction(this));

helpMenu.add( new HelpAction(this));
top.add( fileMenu );
top.add( editMenu );
top.add(helpMenu);

return top;
}
//初始化工具栏
protected ToolBarManager createToolBarManager(int style) {
    ToolBarManager toolBar = new ToolBarManager(style);
    toolBar.add(new OpenAction(this));
    toolBar.add( new SaveAction(this) );
    toolBar.add( new Separator());
    toolBar.add( new PrintAction(this) );

    toolBar.add( new UndoAction(this));
    toolBar.add( new RedoAction(this));
    toolBar.add( new Separator());
    toolBar.add( new SearchAction(this));
    toolBar.add( new Separator());
    toolBar.add( new PreferenceAction(this));
    toolBar.add( new HelpAction(this));
    return toolBar;
}
public PersistentDocument getDocument() {
    return document;
}
public SourceViewer getViewer() {
    return viewer;
}
public EventManager getEventManager() {
    return eventManager;
}
public PreferenceStore getPreference() {
    return preference;
}
public void setPreference(PreferenceStore preference) {
    this.preference = preference;
}
}
```

```

public IUndoManager getUndoManager() {
    return undoManager;
}
public JSEditorConfiguration getConfiguration() {
    return configuration;
}
//为 IPropertyChangeListener 接口中的方法, 当设置首选项的字体时, 重新设置编辑器的字体
public void propertyChange(PropertyChangeEvent event) {
    if (event.getNewValue() != null)
        return;
    if (event.getProperty().equals(Constants.CODE_FONT))
        eventManager.setCodeFont( (FontData[]) event.getNewValue());
}
}

```

21.3.2 主窗口程序代码分析

下面来具体分析一下该程序中所需要注意的问题。

1. TextViewer 对象

在第 19 章中, 介绍过 MVC 模式的树和表格等控件。那么 TextViewer 对象则可以理解为是文本处理的 MVC 对象, TreeView 底层封装了 Tree 控件, 使树的数据和视图分开。而 TextViewer 底层封装的是 StyledText 对象, 用于显示文本, 文本的数据部分则是由实现了 IDocument 接口的对象提供的。简单地说, 要创建一个简单的文本编辑器, 只需要以下所示的代码即可:

```

TextViewer viewer = new TextViewer( shell ,SWT.V_SCROLL | SWT.H_SCROLL);
viewer.setDocument( new Document());

```

而在本程序中使用的是 SourceViewer 对象, 该类继承自 TextViewer, 除了一般的编辑文本的功能外, 增加了编辑源代码的功能, 适用作代码编辑器。另外该类还有一个子类 ProjectionViewer 类, 比 SourceViewer 功能更加丰富, 比如说可以提供折叠代码的效果等。如图 21.2 所示为这几个类的继承关系图。

2. IDocument 接口

处理文本另一个重要的概念就是 IDocument 接口, 它提供了文本的数据, 包括文本的修改、字符的位置、行数等属性和操作。实现该接口常用的有 Document 类和 ProjectionDocument 类, 后者主要是与 ProjectionViewer 一起使用, 通常情况下使用 Document 类就可以了。如图 21.3 所示为实现 IDocument 接口的类及其继承关系图。

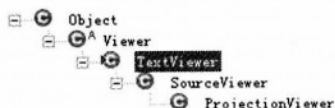


图 21.2 TextViewer 的类继承关系示意图

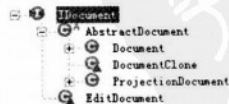


图 21.3 实现 IDocument 接口的类及其继承关系图

本程序中使用的 PersistentDocument 类继承自 Document 类，为文档增加了打开和保存数据的功能。该 PersistentDocument 类实现的具体代码如下：

PersistentDocument.java

```
package com.fengmanfei.ch21;

import java.io.*;
import org.eclipse.jface.text.*;

public class PersistentDocument extends Document implements IDocumentListener {
    private String fileName;//打开的文件名
    private boolean dirty;//文件是否修改过
    public PersistentDocument() {
        this.addDocumentListener(this);//为该文档注册文档监听器
    }

    //保存到指定的文件
    public void save() throws IOException {
        if (fileName == null)
            throw new IllegalStateException("文件名不为空！");
        BufferedWriter out = null;
        try {
            out = new BufferedWriter(new FileWriter(fileName));
            out.write(get());
            dirty = false;
        } finally {
            try {
                if (out != null)
                    out.close();
            } catch (IOException e) {
            }
        }
    }

    //打开文件的方法
    public void open() throws IOException {
        if (fileName == null)
            throw new IllegalStateException("文件名不为空！");
        BufferedReader in = null;
        try {
            in = new BufferedReader(new FileReader(fileName));
            StringBuffer buf = new StringBuffer();
            int n;
            while ((n = in.read()) != -1) {
                buf.append((char) n);
            }
            set(buf.toString());
            dirty = false;
        } finally {
        }
    }
}
```

```
        try {
            if (in != null)
                in.close();
        } catch (IOException e) {
        }
    }
}

public boolean isDirty() {
    return dirty;
}

public void setDirty(boolean dirty) {
    this.dirty = dirty;
}

public String getFileName() {
    return fileName;
}

public void setFileName(String fileName) {
    this.fileName = fileName;
}

//接口中的方法，空实现
public void documentAboutToBeChanged(DocumentEvent arg0) {
}

//接口中的方法，当文档修改后，设置已修改状态
public void documentChanged(DocumentEvent arg0) {
    dirty = true;
}
}
```

该程序很容易理解，为文档增加了读取和保存文件的方法。对文档中的字符的获取是通过父类中的 `get()` 方法获得的，而设置文档字符的方法是通过父类中的 `set()` 方法设置的。

21.3.3 启动主窗口程序

最后，看一下调用主窗口程序类中的代码。程序的入口是通过 `JSEditorTestDrive` 类中的 `main` 方法来调用的，在启动主窗口程序前，要首先加载首选项文件，因为代码着色部分要使用首选项保存文件中的对各种颜色的设置。

该启动程序的代码实现如下：

JSEditorTestDrive.java

```
package com.fengmanfei.ch21;

import java.io.IOException;
import org.eclipse.swt.widgets.Display;

public class JSEditorTestDrive {
    public static void main(String[] args) {
        //创建主窗口对象
```



```

JSEditor jsApp = new JSEditor();
//为主窗口对象设置首选项对象
//首选项对象通过 ResourceManager 中的静态方法获得
jsApp.setPreference(ResourceManager.getPreferenceStore());
//为首选项保存对象注册选项改变监听器
ResourceManager.getPreferenceStore().addPropertyChangeListener(jsApp);
jsApp.setBlockOnOpen(true);
jsApp.open();
Display.getCurrent().dispose();
//窗口关闭后保存设置的首选项
try {
    ResourceManager.getPreferenceStore().save();
} catch (IOException e) {
    e.printStackTrace();
}
}
}

```

21.4 代码着色

下面来看一下如何使 JavaScript 代码进行着色。对代码着色要遵循一定的规则，比如说要区别注释和关键字，也要区别声明的字符和内置的一些对象等，如图 21.4 所示为一段着色后的代码效果图。

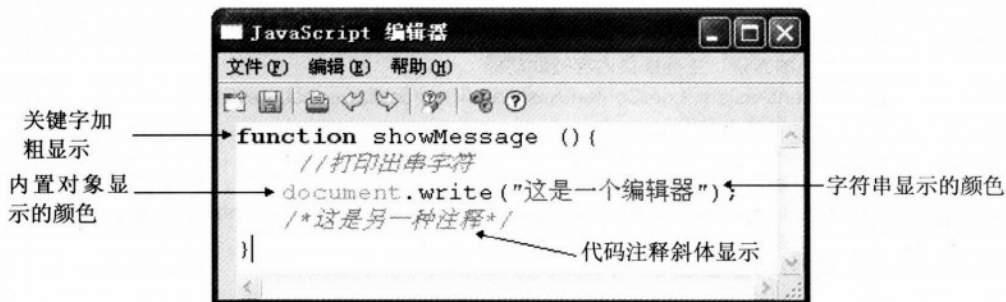


图 21.4 进行着色后的代码效果图

21.4.1 源代码配置类 (SourceViewerConfiguration)

要实现对代码的着色，主要是通过设置 SourceView 对象中的 configure 方法来实现的，可以在主程序窗口的代码中找到以下代码：

```
viewer.configure(configuration); //设置文档配置
```

其中，configuration 是 JSEditorConfiguration 配置对象，对代码实现的着色和内容辅助的功

能都是通过该对象来设置的。JSEditorConfiguration 类继承自 SourceViewerConfiguration 类, 通过覆盖父类中的方法可以实现对代码进行着色和内容辅助等功能。该类具体实现的代码如下:

JSEditorConfiguration.java

```
package com.fengmanfei.ch21;

import org.eclipse.jface.text.IDocument;
import org.eclipse.jface.text.contentassist.*;
import org.eclipse.jface.text.presentation.*;
import org.eclipse.jface.text.rules.*;
import org.eclipse.jface.text.source.*;

public class JSEditorConfiguration extends SourceViewerConfiguration {

    private JSEditor editor ;
    public JSEditorConfiguration( JSEditor editor ){
        this.editor=editor;
    }
    //覆盖父类中的方法, 主要提供代码着色功能
    public IPresentationReconciler getPresentationReconciler(ISourceViewer sourceViewer) {
        PresentationReconciler reconciler = new PresentationReconciler();
        DefaultDamagerRepairer dr = new DefaultDamagerRepairer(new JSCodeScanner());
        reconciler.setDamager(dr, IDocument.DEFAULT_CONTENT_TYPE);
        reconciler.setRepairer(dr, IDocument.DEFAULT_CONTENT_TYPE);
        return reconciler;
    }
    //覆盖父类中的方法, 主要提供内容辅助功能
    public IContentAssistant getContentAssistant(ISourceViewer sourceViewer) {
        //创建内容助手对象
        ContentAssistant contentAssistant = new ContentAssistant();
        //设置提示的内容
        contentAssistant.setContentAssistProcessor(new ObjectContentAssistant(),IDocument.
DEFAULT_CONTENT_TYPE);
        //设置自动激活提示
        contentAssistant.enableAutoActivation(true);
        //设置自动激活提示的时间为 500 毫秒
        contentAssistant.setAutoActivationDelay(500);
        return contentAssistant;
    }
}
```

下面来具体分析一下该程序具体实现的代码, 覆盖父类中的 `getPresentationReconciler` 方法, 可以设置当输入文本时可处理的一些事件。IPresentationReconciler 接口负责处理文件修改的过程, PresentationReconciler 实现了该接口。当文本修改时, 可以认为当前的文本不再正确。此时, 会调用 IPresentationDamager 对象计算被修改文档所在的区域, 而 IPresentationRepairer 对象将利用新的文本属性来进行修改, 所以要使用 `setDamager` 和

setRepairer 方法。

另外 DefaultDamagerRepairer 对象实现了 IPresentationDamager 和 IPresentationRepairer 接口。该对象可以通过设定一个规则进行代码扫描，当扫描的代码符合规则时，就会按照规则指定代码样式修复。

21.4.2 基于规则的代码扫描器类（RuleBasedScanner）

下面看一下具体如何设置代码修复的规则。在以上的程序中找到以下所示的一行代码：

```
DefaultDamagerRepairer dr = new DefaultDamagerRepairer(new JSCodeScanner());
```

可以发现，在以上的代码中创建了一个 JSCodeScanner 对象。该对象即为扫描代码的规则对象。首先来看一下该类的具体代码，然后再进行分析。该类的代码如下：

JSCodeScanner.java

```
package com.fengmanfei.ch21;

import java.util.ArrayList;
import java.util.List;
import org.eclipse.jface.text.TextAttribute;
import org.eclipse.jface.text.rules.*;
import org.eclipse.swt.SWT;

public class JSCodeScanner extends RuleBasedScanner {

    private TextAttribute keywords ;//关键字的文本属性
    private TextAttribute string ;//字符串的文本属性
    private TextAttribute object ;//内置对象的文本属性
    private TextAttribute comment ;//注释部分的文本属性
    public JSCodeScanner(){
        //获得首选项中所设置的颜色，初始化各文本属性
        keywords = new TextAttribute (ResourceManager.getColor(Constants.COLOR_
KEYWORD),null,SWT.BOLD);
        string = new TextAttribute (ResourceManager.getColor(Constants.COLOR_STRING));
        object = new TextAttribute (ResourceManager.getColor(Constants.COLOR_OBJECT));
        comment = new TextAttribute (ResourceManager.getColor(Constants.COLOR_
COMMENT),null,SWT.ITALIC);
        //设置代码的规则
        setupRules();
    }
    private void setupRules() {
        //用一个 List 集合对象保存所有的规则
        List rules = new ArrayList();
        //字符串的规则
        rules.add(new SingleLineRule("\"", "\"",new Token( string), '\'));
        rules.add(new SingleLineRule("'", "'", new Token( string), '\'));
        //注释的规则
```

```

rules.add(new SingleLineRule("/**", "*/", new Token( comment), "\\\"));
rules.add(new EndOfLineRule("//", new Token( comment), "\\\"));
//空格的规则
rules.add(new WhitespaceRule(new IWhitespaceDetector() {
    public boolean isWhitespace(char c) {
        return Character.isWhitespace(c);
    }
}));
//关键字的规则
WordRule keywordRule = new WordRule(new KeywordDetector());
for (int i = 0, n = Constants.JS_SYNTAX_KEYWORD.length; i < n; i++)
    keywordRule.addWord(Constants.JS_SYNTAX_KEYWORD[i], new Token( keywords ));
rules.add(keywordRule);
//内置对象的规则
WordRule objectRule = new WordRule(new ObjectDetector());
for (int i = 0, n = Constants.JS_SYNTAX_BUILDIB_OBJECT.length; i < n; i++)
    objectRule.addWord(Constants.JS_SYNTAX_BUILDIB_OBJECT[i], new Token( object ));
rules.add(objectRule);
//集合类中保存的规则转化为数组
IRule[] result = new IRule[rules.size()];
rules.toArray(result);
//调用父类中的方法，设置规则
//此方法非常重要
setRules(result);
}

```

使用该类方法的主要目的是，为扫描代码提供多个规则。可以这样理解，当修改文本时，程序会自动搜索所有设置的规则，如果当前的文本符合设置的某一个规则时，就按照这个规则所设定的文字样式重新显示文本。所以，对规则的设定直接影响着如何对不同的代码进行不同的着色。

该类 `JSCodeScanner` 继承自 `RuleBasedScanner`，这样就可以使用设置的规则来对代码进行扫描了。

21.4.3 设置代码扫描规则

下面就来详细地看一下 JFace 的文本框架中提供了哪些规则。图 21.5 显示了所有有关规则的类。其中所有的规则都实现了 `IRule` 接口。这里着重讲述一下该程序中使用的规则类。



图 21.5 JFace 中有关规则的类继承关系图

1. SingleLineRule: 单行规则类

SingleLineRule 可以设置指定两个字符间的规则。它的构造方法有:

- ❑ SingleLineRule(String startSequence, String endSequence, IToken token)。

其中 startSequence 表示起始的字符, endSequence 表示终止的字符, token 为符合该规则时所应用的代码样式 IToken 对象。

- ❑ SingleLineRule(String startSequence, String endSequence, IToken token, char escapeCharacter)。

其他参数与上一个构造方法意义相同, 其中 escapeCharacter 表示转义符。

- ❑ SingleLineRule(String startSequence, String endSequence, IToken token, char escapeCharacter, boolean breaksOnEOF)。

其中 breaksOnEOF 表示是否在文件末尾终止该规则。

- ❑ SingleLineRule(String startSequence, String endSequence, IToken token, char escapeCharacter, boolean breaksOnEOF, boolean escapeContinuesLine)。

其中 escapeContinuesLine 表示是否在该行结束后使用转义符。

例如, 在本程序创建该规则对象的代码如下所示:

```
rules.add(new SingleLineRule("/*", "*/", new Token( comment), "\\\\"));
```

表示该规则是在两个字符 “/*” 和 “*/” 之间应用, 也就是本例程序中注释部分的规则。

SingleLineRule 规则的使用很简单。

2. EndOfLineRule: 没有终止字符的规则类

EndOfLineRule 对象可以设置只有初始字符相同后的规则。例如本例程序中, 另一种注释的写法, 也就是带有 “//” 标记后的都可以应用此规则。

例如以下代码所示:

```
rules.add(new EndOfLineRule("//", new Token( comment), "\\\\"));
```

3. WordRule: 单词规则

WordRule 对象应用于对某些特定的字符设置的规则, 一般可以用来设置关键字的规则, 当输入的字符串为规则中所指定的一些字符串时, 就可以使用该对象。与前两种规则相比, 这个规则稍微复杂一些。

首先看一下构造方法:

- ❑ WordRule(IWordDetector detector)。

其中 IWordDetector 为接口, 创建 WordRule 对象时要使用实现这个接口的对象。

- ❑ WordRule(IWordDetector detector, IToken defaultToken)。

也可以设置默认的 IToken 代码格式对象。

本程序中对关键字设置的规则使用的就是 WordRule 规则, 创建的代码如下所示:

```
WordRule keywordRule = new WordRule(new KeywordDetector());
```

这里 KeywordDetector 对象是实现了 IWordDetector 的对象。该类的具体代码如下:

KeywordDetector.java

```
package com.fengmanfei.ch21;

import org.eclipse.jface.text.rules.IWordDetector;
public class KeywordDetector implements IWordDetector {
    // 接口中的方法，字符是否是单词的开始
    public boolean isWordStart(char c) {
        //循环所有的关键字
        //如果有关键字中的第一个字符匹配该字符，则返回 true
        for (int i = 0, n = Constants.JS_SYNTAX_KEYWORD.length; i < n; i++)
            if (c == ((String) Constants.JS_SYNTAX_KEYWORD[i]).charAt(0))
                return true;
        return false;
    }
    // 接口中的方法，字符是否是单词中的一部分
    public boolean isWordPart(char c) {
        //循环所有的关键字
        //如果关键字的字符中有该字符，则返回 true
        for (int i = 0, n = Constants.JS_SYNTAX_KEYWORD.length; i < n; i++)
            if (((String) Constants.JS_SYNTAX_KEYWORD[i]).indexOf(c) != -1)
                return true;
        return false;
    }
}
```

另外，创建 WordRule 对象后，还要使用以下方法将该规则所应用的字符添加到该对象中：

```
addWord(String word, IToken token);
```

word 为该规则应用的字符串，IToken 为规则匹配后所应用的代码样式。例如本程序是循环所有的关键字字符，然后添加到对象中，代码如下：

```
for (int i = 0, n = Constants.JS_SYNTAX_KEYWORD.length; i < n; i++)
    keywordRule.addWord(Constants.JS_SYNTAX_KEYWORD[i], new Token( keywords ));
```

同理，对 JavaScript 内置对象也是通过此规则来设定的。以下只给出了创建内置对象规则时所使用的 ObjectDetector 类的代码：

ObjectDetector.java

```
package com.fengmanfei.ch21;

import org.eclipse.jface.text.rules.IWordDetector;
public class ObjectDetector implements IWordDetector {
    public boolean isWordStart(char c) {
        for (int i = 0, n = Constants.JS_SYNTAX_BUILDIB_OBJECT.length; i < n; i++)
            if (c == ((String) Constants.JS_SYNTAX_BUILDIB_OBJECT[i]).charAt(0))
                return true;
        return false;
    }
}
```



```
public boolean isWordPart(char c) {  
    for (int i = 0, n = Constants.JS_SYNTAX_BUILDIB_OBJECT.length; i < n; i++)  
        if (((String) Constants.JS_SYNTAX_BUILDIB_OBJECT[i]).indexOf(c) != -1)  
            return true;  
    return false;  
}
```

21.4.4 提取类 (Token) 和文本属性类 (TextAttribute)

最后再来看一下规则所使用的 IToken 类, IToken 为一个接口, 实现该接口的类是 Token, 在规则中使用 Token 对象可以将符合该规则字符转换成对应 Token 中设置的文本格式。对于文本格式的设置使用的是 TextAttribute 类。

TextAttribute 类很简单, 有以下两种构造方法:

❑ TextAttribute(Color foreground)。

其中 foreground 为字符显示的前景色。

❑ TextAttribute(Color foreground, Color background, int style)。

其中 background 为字符显示背景色, style 表示样式常量, SWT.BOLD 表示为加粗, SWT.ITALIC 表示倾斜。

最后, 本程序中规则的设置的字体的颜色是从首选项获得的, 具体的代码请参阅 ResourceManager 类相关的代码部分。

21.5 内容辅助

只有代码着色功能还是很简单的编辑器, 根本没有用上 JFace 提供的强大的文本处理。读者在进行编码的过程中, 代码提示的功能可以减轻许多不必要的工作, 大大提高了开发效率。对于本例中的 JavaScript 编辑器, 希望在编码时输入 “.” 时, 能够根据当前输入的对象来提供一些内容的提示, 具体的效果如图 21.6 所示。

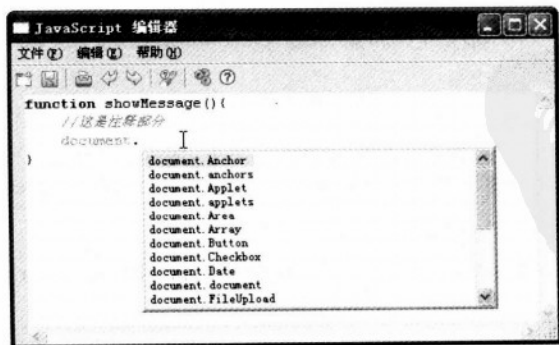


图 21.6 输入 “.” 后出现的内容提示效果图

21.5.1 配置编辑器的内容助手

下面就来仔细分析一下如何实现该功能。请读者在上文中找到 `JEditorConfiguration.java` 的代码部分。找到 `getContentAssistant` 方法，该方法为设置文档的内容辅助的方法。

该方法返回的是一个接口对象 `IContentAssistant`，本例使用的是该接口的一个标准实现 `ContentAssistant` 类。在创建 `ContentAssistant` 对象后要设置该内容辅助的处理器对象，例如以下代码：

```
ContentAssistant contentAssistant = new ContentAssistant();  
//设置提示的内容  
contentAssistant.setContentAssistProcessor(new  
ObjectContentAssistant(),IDocument.DEFAULT_CONTENT_TYPE);
```

其中，`ObjectContentAssistant` 类是需要设置的内容显示的过程器，该类要实现 `IContentAssistProcessor` 接口，所以说，设置辅助内容的过程主要是在 `ObjectContentAssistant` 类中。

21.5.2 内容辅助类

下面具体来看一下该内容辅助类的具体代码，`ObjectContentAssistant` 的实现代码如下：

ObjectContentAssistant.java

```
package com.fengmanfei.ch21;  
  
import java.util.ArrayList;  
import java.util.List;  
import org.eclipse.jface.text.*;  
import org.eclipse.jface.text.contentassist.*;  
  
public class ObjectContentAssistant implements IContentAssistProcessor {  
  
    // 接口中的方法，获得内容的提示数组  
    public ICompletionProposal[] computeCompletionProposals(ITextViewer viewer,  
        int offset) {  
        IDocument doc = viewer.getDocument();  
        //获得提示列表中的内容  
        List list = computeObject(getObjectName(doc, offset), offset);  
        return (CompletionProposal[]) list.toArray(new CompletionProposal[list.size()]);  
    }  
    // 接口中的方法，空实现  
    public IContextInformation[] computeContextInformation(ITextViewer viewer,  
        int offset) {  
        return null;  
    }  
    // 接口中的方法，设置何时激活提示，这里当输入"."时激活内容提示
```

```
public char[] getCompletionProposalAutoActivationCharacters() {
    return new char[] { '.' };
}
// 接口中的方法
public char[] getContextInformationAutoActivationCharacters() {
    return null;
}
// 接口中的方法
public String getErrorMessage() {
    return null;
}
// 接口中的方法
public IContextInformationValidator getContextInformationValidator() {
    return null;
}
//获得提示时之前输入的字符
public String getObjectNames(IDocument doc, int offset) {
    //offset 为当前光标所在位置, 也就是偏移量
    StringBuffer buf = new StringBuffer();
    offset--;
    //依次从当前位置查找, 直到字符串为空格或者是 "." 时停止, 然后将字符串反转
    while (true) {
        try {
            char c = doc.getChar(--offset); //获得前一个字符
            if (Character.isWhitespace(c)) //如果为空格, 跳出
                break;
            if (c == '.') //如果为 "." 则跳出
                break;
            buf.append(c);
        } catch (BadLocationException e) {
            break;
        }
    }
    return buf.reverse().toString(); //最后将字符反转
}
// 设置内容提示中的内容
public List computeObjectNames(String objName, int offset) {
    //objName 首先看 objName 是否为内置的对象
    List list = new ArrayList();
    boolean bFind = false;
    for (int i = 0; i < Constants.JS_SYNTAX_BUILDIB_OBJECT.length; i++) {
        String tempString = Constants.JS_SYNTAX_BUILDIB_OBJECT[i];
        if (objName.equals(tempString))
        {
            bFind = true;
            break;
        }
    }
    //如果是内置对象, 则将所有对象的字符添加到内容提示列表中
```

```
if (bFind) {
    for (int i = 0; i < Constants.JS_SYNTAX_BUILDIB_OBJECT.length; i++) {
        String insert = objName + "." + Constants.JS_SYNTAX_BUILDIB_OBJECT[i];
        //CompletionProposal 对象
        CompletionProposal proposal = new CompletionProposal(insert,
            offset - objName.length() - 1, objName.length()+1, insert.length()+1);
        list.add(proposal);
    }
}
return list;
}
```

下面来仔细分析一下该程序实现的原理。ObjectContentAssistant 实现了该接口，本程序中实现了其中的 getCompletionProposalAutoActivationCharacters 和 computeCompletion Proposals 两个方法。其中前者的方法很简单，设置当输入为何字符串时激活内容辅助的方法；后者返回的是一个 ICompletionProposal 数组列表，也就是显示内容辅助中的列表内容。

21.5.3 辅助建议类（CompletionProposal）

ICompletionProposal 为一个接口，实现了该接口的类如图 21.7 所示。本例中使用的是 CompletionProposal 类来返回的辅助内容列表。

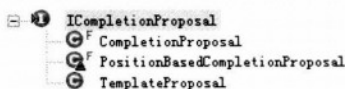


图 21.7 实现了 ICompletionProposal 接口的类

一般情况下，并不是只要输入“.”就出现内容提示，通常输入“.”时要获得该对象的属性。所以要获得当前输入的“.”之前的字符串信息，本程序中获得输入“.”之前的字符串的方法是 getObjectNames。该方法从当前的位置向前循环，当遇到空格或是“.”时，结束循环，获得输入“.”位置到结束位置的字符。

例如输入“document.form.”后，输入“.”此时通过 getObjectNames 方法获得的字符串为“form”，这样就可以判断是否是内置的对象了。

判断是否为内置对象字符的功能是在 computeObject 方法中实现的。该方法首先循环所有的内置对象的字符，查找是否存在通过 getObjectNames 获得属性。如果存在，则将内置对象的字符添加到 CompletionProposal 对象中，最后返回一个集合对象。

下面具体看一下 CompletionProposal 类构造方法，如下所示：

❑ CompletionProposal(String replacementString,int replacementOffset,int replacementLength,int cursorPosition)。

其中，replacementString 为选中提示内容的字符替换的字符串；replacementOffset 为替换文本所在文档的位置，也就是偏移量；replacementLength 为替换文本的字符长度，插入提示内容后鼠标所在位置。

- ❑ `CompletionProposal(String replacementString, int replacementOffset, int replacementLength, int cursorPosition, Image image, String displayString, IContextInformation contextInformation, String additionalProposalInfo)`。

其中, `image` 为内容提示中显示的图标; `displayString` 为内容提示中显示的文本, 但并不是选中后替换的文本; `contextInformation` 为当光标放在提示列表中的一项时所出现的具体的提示信息; `additionalProposalInfo` 为附加的提示信息。

当然, 本程序的内容辅助功能还非常简单, 并未实现动态地获得辅助的内容, 读者有兴趣可以继续完善该内容辅助的功能。

21.6 文档的撤销与重复

JFace 文本处理框架中, 使用 `IUndoManager` 对象来管理文档的撤销与恢复, `IUndoManager` 为一个接口, 实现该接口的类为 `DefaultUndoManager`。

21.6.1 文档管理器对象 (DefaultUndoManager)

使用 `DefaultUndoManager` 对象很简单, 如以下代码所示为文本编辑对象创建了文档撤销管理器的方法。

```
//初始化撤销管理器对象, 默认可撤销 100 次
IUndoManager undoManager = new DefaultUndoManager(100);
undoManager.connect(viewer); //应用于文档
```

将管理器应用到 `TextViewer` 对象后, 就可以使用 `undoManager.redo()` 方法来恢复操作, 使用 `undoManager.undo()` 撤销操作。

本程序中, 文档的撤销和恢复是通过菜单和工具栏操作的, 如图 21.8 所示为编辑菜单的效果图。

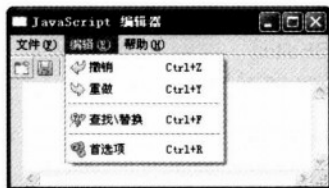


图 21.8 编辑菜单的效果图

21.6.2 撤销操作的实现

撤销和重做的功能是通过 `UndoAction` 和 `RedoAction` 两个类实现的。这两个类与之前学习创建菜单的方法相同, 都继承自 `Action` 类。主要是注意处理事件部分的代码。

撤销操作 UndoAction 类实现的代码如下：

UndoAction.java

```
package com.fengmanfei.ch21.action;

import org.eclipse.jface.action.Action;
import org.eclipse.jface.resource.ImageDescriptor;
import com.fengmanfei.ch21.JSEditor;
import com.fengmanfei.ch21.ResourceManager;

public class UndoAction extends Action {
    private JSEditor editor;
    public UndoAction(JSEditor editor) {
        super("撤销@Ctrl+Z");

        this.setImageDescriptor(ImageDescriptor.createFromFile(ResourceManager.class, "icons\\undo.gif"));
        this.editor = editor;
    }
    public void run() {
        editor.getUndoManager().undo();
    }
}
```

21.6.3 恢复操作的实现

恢复操作 RedoAction 类实现的代码如下：

RedoAction.java

```
package com.fengmanfei.ch21.action;

import org.eclipse.jface.action.Action;
import org.eclipse.jface.resource.ImageDescriptor;
import com.fengmanfei.ch21.JSEditor;
import com.fengmanfei.ch21.ResourceManager;

public class RedoAction extends Action {
    private JSEditor editor;
    public RedoAction(JSEditor editor) {
        super("恢复@Ctrl+Y");

        this.setImageDescriptor(ImageDescriptor.createFromFile(ResourceManager.class, "icons\\redo.gif"));
        this.editor = editor;
    }
    public void run() {
        editor.getUndoManager().redo();
    }
}
```

```

    }
}

```

以上两个程序需要注意的是，创建对象时都要将 `JSEditor` 作为构造参数传入，这样可以方便地获得主窗口对象中的属性。其他类中也是这样处理的。

21.7 查找与替换窗口

对于 `Document` 对象，已经封装了对文本查找和替换的常用操作，要实现查找和替换的功能，只需使用它内置的一些方法即可。所以实现起来比较容易，首先看一下“查找 / 替换”对话框的界面设计。选择“查找 / 替换”命令，将弹出“查找 / 替换”对话框，如图 21.9 所示。

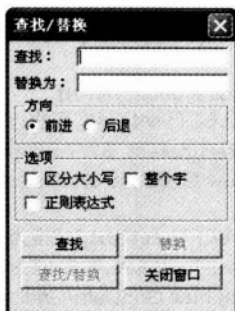


图 21.9 “查找 / 替换”对话框

21.7.1 窗口的界面设计

该对话框整体的布局是网格格式布局，并且为两列。中间两个为 `Group` 分组面板，布局也为网格格式布局。该界面的具体代码如下：

FindAndReplace.java

```

package com.fengmanfei.ch21.dialog;

import org.eclipse.jface.dialogs.Dialog;
import org.eclipse.jface.text.FindReplaceDocumentAdapter;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.*;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.widgets.*;

import com.fengmanfei.ch21.JSEditor;

public class FindAndReplace extends Dialog {
    private JSEditor editor;
    private Button btFind;// “查找” 按钮
    private Button btReplace;// “替换” 按钮
}

```

```
private Button btFindAndReplace;// “查找/替换” 按钮
private Button btClose;// “关闭” 按钮

private FindReplaceDocumentAdapter findDdapter;
public FindAndReplace(JSEditor editor ,Shell parentShell) {
    super(parentShell);
    this.editor = editor ;
    //查找文档字符的适配器对象
    findDdapter = new FindReplaceDocumentAdapter(this.editor.getDocument());
}
protected void configureShell(Shell newShell) {
    super.configureShell(newShell);
    newShell.setText("查找/替换");
    newShell.setSize(200,270);
}
protected Control createContents(Composite parent) {
    //创建对话框的控件
    parent.setLayout(new GridLayout(2, false));
    new Label( parent , SWT.LEFT).setText("查找: ");
    final Text findText = new Text(parent, SWT.BORDER);
    findText.setLayoutData( new GridData(GridData.FILL_HORIZONTAL));

    new Label( parent , SWT.LEFT).setText("替换为: ");
    final Text replaceText = new Text(parent, SWT.BORDER);
    replaceText.setLayoutData( new GridData(GridData.FILL_HORIZONTAL));

    Group group = new Group( parent, SWT.NONE);
    GridData data = new GridData(GridData.FILL_HORIZONTAL);
    data.horizontalSpan = 2;
    group.setLayoutData(data);
    group.setText("方向");
    group.setLayout( new GridLayout(2,true) );

    final Button forwardButton = new Button(group,SWT.RADIO);
    forwardButton.setText("前进");

    final Button backButton = new Button(group,SWT.RADIO);
    backButton.setText("后退");

    group = new Group( parent, SWT.NONE);
    data = new GridData(GridData.FILL_HORIZONTAL);
    data.horizontalSpan = 2;
    group.setLayoutData(data);
    group.setText("选项");
    group.setLayout( new GridLayout(2,true) );

    final Button match = new Button(group,SWT.CHECK);
    match.setText("区分大小写");
```

```

final Button wholeWord = new Button(group, SWT.CHECK);
wholeWord.setText("整个字");

final Button regexp = new Button(group, SWT.CHECK);
regexp.setText("正则表达式");

Composite composite = new Composite( parent , SWT.NONE);
data = new GridData(GridData.FILL_HORIZONTAL);
data.horizontalSpan = 2;
composite.setLayoutData(data);
composite.setLayout( new GridLayout(2,true));

btFind = new Button( composite , SWT.PUSH );
btFind.setText("查找");
btFind.setLayoutData( new GridData(GridData.FILL_HORIZONTAL));

btReplace = new Button( composite , SWT.PUSH );
btReplace.setText("替换");
btReplace.setLayoutData( new GridData(GridData.FILL_HORIZONTAL));

btFindAndReplace = new Button( composite , SWT.PUSH );
btFindAndReplace.setText("查找/替换");
btFindAndReplace.setLayoutData( new GridData(GridData.FILL_HORIZONTAL));

btClose = new Button( composite , SWT.PUSH );
btClose.setText("关闭窗口");
btClose.setLayoutData( new GridData(GridData.FILL_HORIZONTAL));
//设置按钮的事件
//设置查找选项时，正则表达式与匹配整个字符不能同时使用
wholeWord.addSelectionListener( new SelectionAdapter(){
    public void widgetSelected(SelectionEvent event) {
        if ( wholeWord.getSelection() ){
            regexp.setSelection( false );
            regexp.setEnabled( false );
        }else{
            regexp.setEnabled( true );
        }
    }
});
regexp.addSelectionListener( new SelectionAdapter(){
    public void widgetSelected(SelectionEvent event) {
        if ( regexp.getSelection() ){
            wholeWord.setSelection( false );
            wholeWord.setEnabled( false );
        }else{
            wholeWord.setEnabled( true );
        }
    }
});

```



```

//为“查找”按钮注册事件监听器
btFind.addSelectionListener(new SelectionAdapter() {
    public void widgetSelected(SelectionEvent event) {
        boolean b = editor.getEventManager().isFind( findDdapater,
            findText.getText(),forwardButton.getSelection(),match.getSelection(),
            wholeWord.getSelection(),regexp.getSelection());
        //如果找到匹配的字符，将“替换”和“替换全部”按钮设置为可用状态
        enableReplaceButtons(b);
    }
});
//为“替换”按钮注册事件监听器
btReplace.addSelectionListener(new SelectionAdapter() {
    public void widgetSelected(SelectionEvent event) {
        editor.getEventManager().doReplace( findDdapater , replaceText.getText());
        enableReplaceButtons(false);
    }
});
//为“查找/替换”按钮注册事件监听器
btFindAndReplace.addSelectionListener(new SelectionAdapter() {
    public void widgetSelected(SelectionEvent event) {
        editor.getEventManager().doReplace( findDdapater , replaceText.getText());
        boolean b = editor.getEventManager().isFind( findDdapater,
            findText.getText(),forwardButton.getSelection(),match.getSelection(),
            wholeWord.getSelection(),regexp.getSelection());
        enableReplaceButtons(b);
    }
});
//为“关闭窗口”按钮注册监听器
btClose.addSelectionListener(new SelectionAdapter() {
    public void widgetSelected(SelectionEvent event) {
        getShell().close();
    }
});
//设置向前为默认选中状态
forwardButton.setSelection(true);
//初始化时“替换”和“查找/替换”按钮不可用
enableReplaceButtons(false);
//设置焦点为查找的文本框
findText.setFocus();
return parent;
}
private void enableReplaceButtons(boolean enable) {
    btReplace.setEnabled(enable);
    btFindAndReplace.setEnabled(enable);
}
}

```

该程序的代码虽然很长，但无非是使用了最基础的 SWT 布局的知识，读者应该不陌生。关键是要看一下如何处理查找和替换匹配字符的操作。

21.7.2 查找功能的实现

当简单查找按钮后，会调用 `EventManager` 对象的 `isFind` 方法进行查找。`EventManager` 对象是事件管理器对象，集中处理一些事件。另外，需要注意的是，该类中有一个 `FindReplaceDocumentAdapter` 对象，在构造方法中创建的，对文档中字符串的查找主要是通过该对象来实现的。

以下为 `EventManager` 类中 `isFind` 方法具体实现的代码：

```
public boolean isFind(FindReplaceDocumentAdapter adapter, String find,
                    boolean forward, boolean matchCase, boolean wholeWord,
                    boolean regexp) {
    boolean bFind = false;
    IRegion region = null;
    try {
        // 获得当前文本所在的位置，也就是偏移量
        int offset = editor.getViewer().getTextWidget().getCaretOffset();

        if (!forward) {
            Point pt = editor.getViewer().getSelectedRange();
            if (pt.x != pt.y) {
                offset = pt.x - 1;
            }
        }
        // 确保没有超出 adapter 的范围
        if (offset >= adapter.length())
            offset = adapter.length() - 1;
        if (offset < 0)
            offset = 0;
        // 查找字符
        region = adapter.find(offset, find, forward, matchCase, wholeWord, regexp);
        // 如果找到，设置匹配的字符选中，并返回 true
        if (region != null) {
            editor.getViewer().setSelectedRange(region.getOffset(), region.getLength());
            bFind = true;
        }
    } catch (BadLocationException e) {
        ;
    } catch (PatternSyntaxException e) {
        ;
    }
    return bFind;
}
```

21.7.3 替换功能的实现

另外，对文本的替换也是使用 `EventManager` 类中的 `doReplace` 方法，该方法具体实现的代码如下：

```
public void doReplace(FindReplaceDocumentAdapter adapter, String replaceText) {  
    try {  
        adapter.replace(replaceText, false);  
    } catch (BadLocationException e) {  
  
    }  
}
```

最后看一下“编辑”菜单中无打开“查找/替换”对话框对应的事件处理类，该事件是由 `SearchAction` 类来处理的，该类实现的代码如下：

SearchAction.java

```
package com.fengmanfei.ch21.action;  
  
import org.eclipse.jface.action.Action;  
import org.eclipse.jface.resource.ImageDescriptor;  
import com.fengmanfei.ch21.JSEditor;  
import com.fengmanfei.ch21.ResourceManager;  
import com.fengmanfei.ch21.dialog.FindAndReplace;  
  
public class SearchAction extends Action {  
    private JSEditor editor;  
    public SearchAction(JSEditor editor) {  
        super("查找/替换@Ctrl+F");  
  
        this.setImageDescriptor(ImageDescriptor.createFromFile(ResourceManager.class, "icons\\  
search.gif"));  
        this.editor = editor;  
    }  
    public void run() {  
        new FindAndReplace( editor , editor.getShell()).open();  
    }  
}
```

21.8 首选项的对话框

对各种字体关键字的颜色是可以让用户自定义的，在第 18 章中已经学习了有关首选项的知识，这里就不多作详细介绍了。

首先要有一个首选项页面，该页面运行后的效果如图 21.10 所示。

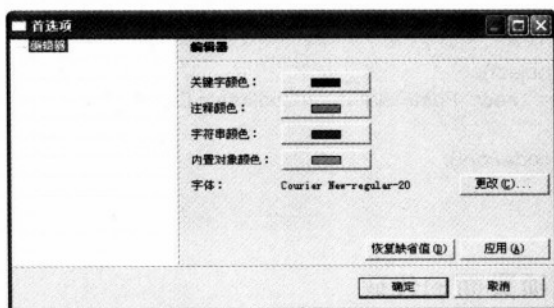


图 21.10 首选项页面效果图

21.8.1 首选项页面的代码实现

该首选项页面的代码如下：

JSPreferencePage.java

```
package com.fengmanfei.ch21;

import org.eclipse.jface.preference.ColorFieldEditor;
import org.eclipse.jface.preference.FieldEditorPreferencePage;
import org.eclipse.jface.preference.FontFieldEditor;

public class JSPreferencePage extends FieldEditorPreferencePage{

    private ColorFieldEditor keyword;
    private ColorFieldEditor comment;
    private ColorFieldEditor string;
    private ColorFieldEditor object;
    private FontFieldEditor codeFont;
    public JSPreferencePage() {
        super(GRID);
    }

    protected void createFieldEditors() {
        keyword = new ColorFieldEditor(Constants.COLOR_KEYWORD,"关键字颜色: ",getField
EditorParent());
        addField(keyword);
        comment = new ColorFieldEditor(Constants.COLOR_COMMENT,"注释颜色: ",getField
EditorParent());
        addField(comment);
        string = new ColorFieldEditor(Constants.COLOR_STRING,"字符串颜色: ",getField
EditorParent());
        addField(string);
```

```
        object = new ColorFieldEditor(Constants.COLOR_OBJECT,"内置对象颜色: ",getFieldEditorParent());
        addField(object);
        codeFont= new FontFieldEditor(Constants.CODE_FONT,"字体: ",getFieldEditorParent());
        addField(codeFont);
    }
}
```

21.8.2 打开首选项页面的代码

最后看一下打开首选项对话框中的事件如何处理，打开首选项的事件处理类是 PreferenceAction。以下为该类实现的具体代码：

PreferenceAction.java

```
package com.fengmanfei.ch21.action;

import org.eclipse.jface.action.Action;
import org.eclipse.jface.preference.PreferenceDialog;
import org.eclipse.jface.preference.PreferenceManager;
import org.eclipse.jface.preference.PreferenceNode;
import org.eclipse.jface.resource.ImageDescriptor;
import com.fengmanfei.ch21.JSEditor;
import com.fengmanfei.ch21.JSPreferencePage;
import com.fengmanfei.ch21.ResourceManager;

public class PreferenceAction extends Action {
    private JSEditor editor;
    public PreferenceAction(JSEditor editor) {
        super("首选项@Ctrl+R");

        this.setImageDescriptor(ImageDescriptor.createFromFile(ResourceManager.class,"icons\\prefs.gif"));
        this.editor = editor;
    }
    public void run() {
        PreferenceManager mgr = new PreferenceManager();
        mgr.addToRoot(new PreferenceNode("edit", "编辑器", null,JSPreferencePage.class.getName()));
        PreferenceDialog dlg = new PreferenceDialog(editor.getShell(), mgr);
        dlg.setPreferenceStore(editor.getPreference());
        dlg.open();
    }
}
```

21.9 文件的打开、保存与打印

在文件菜单栏中提供了对打开文件、保存文件和删除文件的功能，如图 21.11 所示即为菜单栏中“文件”菜单项的效果图。

下面具体看一下这 3 个事件处理类。

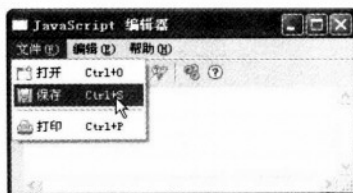


图 21.11 “文件”菜单项

21.9.1 打开文件

打开操作类 `OpenAction` 类实现的代码如下：

`OpenAction.java`

```
package com.fengmanfei.ch21.action;

import org.eclipse.jface.action.Action;
import org.eclipse.jface.resource.ImageDescriptor;
import com.fengmanfei.ch21.*;
import com.fengmanfei.ch21.JSEditor;

public class OpenAction extends Action {
    private JSEditor editor;
    public OpenAction(JSEditor editor) {
        super("打开@Ctrl+O");

        this.setImageDescriptor(ImageDescriptor.createFromFile(ResourceManager.class, "icons\\open.gif"));
        this.editor = editor;
    }
    public void run() {
        editor.getEventManager().openFile();
    }
}
```

其中，打开文件是 `EventManager` 对象中的 `openFile` 方法来处理的，该方法具体实现的代码请读者查看下文中的事件管理类的代码。

21.9.2 保存文件

保存操作类 `SaveAction` 类实现的代码如下：

`SaveAction.java`

```
package com.fengmanfei.ch21.action;
```



```
import org.eclipse.jface.action.Action;
import org.eclipse.jface.resource.ImageDescriptor;
import com.fengmanfei.ch21.JSEditor;
import com.fengmanfei.ch21.ResourceManager;

public class SaveAction extends Action {
    private JSEditor editor;
    public SaveAction(JSEditor editor) {
        super("保存@Ctrl+S");

        this.setImageDescriptor(ImageDescriptor.createFromFile(ResourceManager.class, "icons\\
save.gif"));
        this.editor = editor;
    }
    public void run() {
        editor.getEventManager().saveFile();
    }
}
```

21.9.3 打印文件

打印操作类 PrintAction 类实现的代码如下：

PrintAction.java

```
package com.fengmanfei.ch21.action;

import com.fengmanfei.ch21.JSEditor;
import com.fengmanfei.ch21.ResourceManager;
import org.eclipse.jface.action.Action;
import org.eclipse.jface.resource.ImageDescriptor;

public class PrintAction extends Action{
    private JSEditor editor;
    public PrintAction(JSEditor editor){
        super("打印@Ctrl+P");

        this.setImageDescriptor(ImageDescriptor.createFromFile(ResourceManager.class, "icons\\
print.gif"));
        this.editor = editor;
    }
    public void run() {
        editor.getViewer().getTextWidget().print();
    }
}
```

其中，StyledText 对象提供了打印的方法 print()，可以直接使用该方法进行打印。

21.10 帮助对话框

为了使该 JavaScript 编辑器更加完善, 也增加了帮助窗口, 显示了版本信息。如图 21.12 所示为帮助窗口的对话框运行后的效果图。实现该对话框的代码很简单, 具体如下:

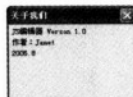


图 21.12 帮助对话框

AboutDialog.java

```
package com.fengmanfei.ch21.dialog;

import org.eclipse.jface.dialogs.Dialog;
import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.*;

public class AboutDialog extends Dialog {

    public AboutDialog(Shell parentShell) {
        super(parentShell);
    }

    protected Control createContents(Composite parent) {
        this.getShell().setSize(200,150);
        this.getShell().setText("关于我们");
        parent.setLayout( new GridLayout());
        new Label(parent, SWT.CENTER).setText("JS 编辑器 Version 1.0");
        new Label(parent, SWT.CENTER).setText("作者: Janet");
        new Label(parent, SWT.RIGHT).setText("2006.8");
        return parent;
    }
}
```

另外, 还要看一下调用该窗口中的事件处理类 HelpAction。该类的代码如下:

HelpAction.java

```
package com.fengmanfei.ch21.action;

import org.eclipse.jface.action.Action;
import org.eclipse.jface.resource.ImageDescriptor;
import com.fengmanfei.ch21.JSEditor;
import com.fengmanfei.ch21.ResourceManager;
import com.fengmanfei.ch21.dialog.AboutDialog;

public class HelpAction extends Action {
    private JSEditor editor;
    public HelpAction(JSEditor editor) {
```

```
        super("帮助@Ctrl+O");

        this.setImageDescriptor(ImageDescriptor.createFromFile(ResourceManager.class, "icons\\help.gif"));
        this.editor = editor;
    }
    public void run() {
        AboutDialog dlg = new AboutDialog( editor.getShell());
        dlg.open();
    }
}
```

21.11 其他的一些工具类

21.11.1 事件管理类

一般在设计系统时，通常会将系统中事件处理集中到一个对象中进行，这样可以管理多个事件。这样做的好处是，一旦有代码需要修改时，只需要修改事件管理类中的代码即可，维护起来比较容易。基于这种设计的思路，本系统中使用 `EventManager` 类实现集中处理事件。该类实现的代码如下：

EventManager.java

```
package com.fengmanfei.ch21;

import java.io.IOException;
import java.util.regex.PatternSyntaxException;
import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.jface.text.*;
import org.eclipse.swt.SWT;
import org.eclipse.swt.graphics.*;
import org.eclipse.swt.widgets.FileDialog;

public class EventManager {
    private JSEditor editor;

    public EventManager(JSEditor editor) {
        this.editor = editor;
    }
    //设置字体
    public void setCodeFont(FontData[] fontData) {
        Font font = editor.getViewer().getTextWidget().getFont();
        if (font != null)
            font.dispose();
        font = new Font(editor.getShell().getDisplay(), fontData);
        editor.getViewer().getTextWidget().setFont(font);
    }
}
```

```
}
//打开文件
public void openFile() {
    FileDialog dialog = new FileDialog(editor.getShell(), SWT.OPEN);
    dialog.setFilterExtensions(new String[] { "*.js", "*.html", "*.htm", "**.*" });
    String name = dialog.open();
    if ((name == null) || (name.length() == 0))
        return;
    try {
        editor.getDocument().setFileName(name);
        editor.getDocument().open();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
//保存文件
public void saveFile() {
    if (!editor.getDocument().isDirty())
        return;
    boolean b = MessageDialog.openConfirm(editor.getShell(), "确认保存",
        "您确实要保存文件吗? ");
    if (b) {
        try {
            editor.getDocument().save();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
//以下两个方法在上文已经显示过代码，这里省略
public boolean isFind()
public void doReplace()
}
```

21.11.2 资源管理类

需要集中管理的除了事件外，还有系统所使用的一些资源。这里使用 `ResourceManager` 对象来集中管理系统中使用的资源，该类使用了 `Singleton` 设计模式，并提供了一些静态方法。该类的具体代码如下：

ResourceManager.java

```
package com.fengmanfei.ch21;

import java.io.IOException;
import org.eclipse.jface.preference.PreferenceStore;
import org.eclipse.jface.resource.ColorRegistry;
import org.eclipse.jface.resource.StringConverter;
```



```
import org.eclipse.swt.graphics.Color;

public class ResourceManager {

    private ResourceManager(){}//设置为 private，不允许创建对象
    private static ColorRegistry colorRegistry;//管理颜色的对象
    //获得颜色注册器对象
    public static ColorRegistry getColorRegistry() {
        if (colorRegistry == null) {
            colorRegistry = new ColorRegistry();
            initColor();
        }
        return colorRegistry;
    }
    //获得颜色注册器中的颜色对象的工具方法
    public static Color getColor( String key ) {
        Color color = getColorRegistry().get(key);
        return color;
    }
    //初始化首选项文件中设置的各种代码的颜色
    private static void initColor() {

        colorRegistry.put(Constants.COLOR_COMMENT,StringConverter.asRGB(getPreferenceStore()
        ().getString(Constants.COLOR_COMMENT)));

        colorRegistry.put(Constants.COLOR_KEYWORD,StringConverter.asRGB(getPreferenceStore()
        ().getString(Constants.COLOR_KEYWORD)));

        colorRegistry.put(Constants.COLOR_STRING,StringConverter.asRGB(getPreferenceStore()
        ().getString(Constants.COLOR_STRING)));

        colorRegistry.put(Constants.COLOR_OBJECT,StringConverter.asRGB(getPreferenceStore()
        ().getString(Constants.COLOR_OBJECT)));
    }
    //保存的首选项设置文件
    private static PreferenceStore preference ;
    //加载首选项文件
    public static PreferenceStore getPreferenceStore() {
        if (preference == null) {
            preference = new PreferenceStore( "C:\\jsEditor.properties");
            try {
                preference.load();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        return preference;
    }
}
```


21.11.3 程序中使用的常量

另外，本系统中将所用到的常量都保存在 Constants 类中。该类中保存的常量具体如下：

Constants.java

```
package com.fengmanfei.ch21;

public class Constants {

    //颜色的 key 值
    public static final String COLOR_COMMENT = "Comment";
    public static final String COLOR_KEYWORD = "Keyword";
    public static final String COLOR_STRING = "String";
    public static final String COLOR_OBJECT = "Object";
    //字体的 key 值
    public static final String CODE_FONT = "Font";
    //语法使用的关键字
    public static final String[] JS_SYNTAX_KEYWORD = {
        "abstract", "boolean", "break", "byte", "case",
        "catch", "char", "class", "const", "continue",
        "default", "delete", "do", "double", "else",
        "extends", "false", "final", "finally", "float",
        "for", "function", "goto", "if", "implements",
        "import", "short", "in", "instanceof", "int",
        "interface", "long", "native", "new", "null",
        "package", "private", "protected", "public", "return",
        "short", "static", "super", "switch", "synchronized",
        "this", "throw", "throws", "transient", "true",
        "try", "typeof", "var", "void", "while",
        "with", "script", "language"
    };
    //语法使用的内置对象
    public static final String[] JS_SYNTAX_BUILDIB_OBJECT = {
        "Anchor", "anchors", "Applet", "applets", "Area",
        "Array", "Button", "Checkbox", "Date", "document",
        "FileUpload", "Form", "forms", "Frame", "frames",
        "Hidden", "history", "Image", "images", "Link", "write"
    };
    //程序中使用的图片常量
    public static final String ICON_OPEN = "icon_open";
    public static final String ICON_SAVE = "icon_save";
    public static final String ICON_PINT = "icon_print";
    public static final String ICON_UNDO = "icon_undo";
    public static final String ICON_REDO = "icon_redo";
    public static final String ICON_SEARCH = "icon_search";
    public static final String ICON_PREFS = "icon_prefs";
    public static final String ICON_HELP = "icon_help";
}
```

21.12 本章小结

本章通过一个完整的案例学习了 JFace 框架中有关文本处理的一些使用,主要有代码着色、内容辅助、撤销与恢复操作、查找和替换操作等。另外程序中也涉及了前几章中的一些知识,也是对前几章的回顾。

当然,这里只是使用了 JFace 强大文本处理框架中的一点东西,如果读者对这方面有兴趣,可以研究一下 JFace 中有关编辑器部分的知识。



第 5 篇



DESIGN

RCP 应用篇

第 22 章 富客户端平台 (RCP) 应用

第 23 章 RCP 开发

设计
PDG

第 22 章 富客户端平台（RCP）应用

RCP（Rich Client Platform）富客户端平台是基于 Eclipse 插件开发的一种应用。它是 Eclipse 3.0 版本后新增的一项功能。通过 RCP 可以快速构建应用程序，具有广阔的应用前景。

22.1 RCP 概述

在 Eclipse 平台的发展过程中，倡导的是插件的思想。可以说，插件是 Eclipse 平台的核心内容，但所有的这些插件的运行都要依赖于 Eclipse 平台的存在。但当程序员开发桌面的应用时，往往想摆脱对 Eclipse IDE 的依赖，而是希望使用最小的运行环境来运行系统。所以在 Eclipse 3.0 以后的版本中逐步地将插件的运行从 Eclipse 的运行平台中剥离出来，从而形成了 RCP。

简单地讲，RCP 系统本质上是 Eclipse 的插件，但运行时却能够脱离 Eclipse 平台而独立运行。这就使得 RCP 的应用更加灵活和广泛。

22.1.1 什么是 RCP

RCP 本质上是 Eclipse 的插件，所以当开发 RCP 应用程序时，可以利用 Eclipse 平台 UI 外观和框架来快速地进行开发。例如创建一个菜单栏、工具栏，在 RCP 开发中很容易，只需要作一定的配置后，编写简单的代码就可以实现复杂的功能，这样就避免了许多重复性的工作。

RCP 的系统可以脱离 Eclipse 平台独立运行，这样大大减少了打包程序后文件的体积，使系统更加小巧和雅观。图 22.1 清晰地显示了 RCP 与 Eclipse 的关系。

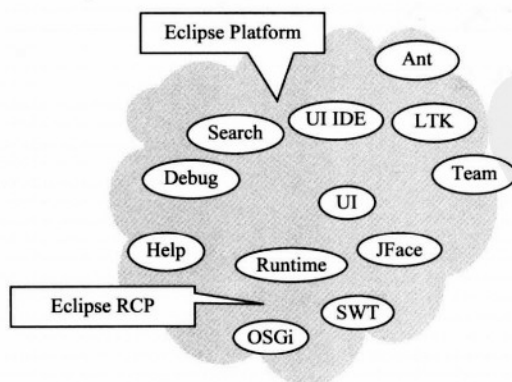


图 22.1 RCP 与 Eclipse 的关系图

另外, 最重要的是 Eclipse 是一个开源平台, 所以采用 RCP 也可以极大地降低系统的成本, 具有很高的商业价值。

22.1.2 RCP 应用的现状

RCP 在 Java 桌面应用有很广阔的前景, 就目前来说, 已经进行了广泛的应用。主要分为两大阵营, 开源的和商业的。

1. 开源的应用

- ❑ Azureus: <http://azureus.sourceforge.net/>, BitTorrent 下载客户端软件, 以支持 40 种语言, 功能强大, 已经被上百万的用户下载使用, 并且获得了 2006 年 Sourceforge.net 的 Best Overall Winner 大奖。
- ❑ Bioclipse: <http://www.bioclipse.net/index.php>, 生物信息学的一个绘图软件, 可以绘制出 DNA 的 3D 图形, 有兴趣的读者可以下载下来具体看一下。
- ❑ BrainBox: <http://eclipsetrader.sourceforge.net/>, 基于 RCP 的股票交易软件, 可以实时地查看股票信息和股票的历史走势图等。
- ❑ jCommander: <http://jcommander.sourceforge.net/>, 基于选项卡界面效果的文件系统管理软件, 可轻松地实现文件的管理。
- ❑ jFire: <http://jfire.org/>, 基于 J2EE 的 ERP 软件, 包括产品管理、客户管理、用户管理等模块, 具有很灵活的自定义功能。另外, 还使用了 GEF 和 BIRT 构建丰富的报表图形系统。还提供了扩展点, 允许用户在此基础上进行开发。
- ❑ jLibrary: <http://jlibrary.sourceforge.net/>, 开源的文档管理系统 (DMS), 可以将普通文件、视频文件和其他类型的文件进行分类, 也可以进行查找和分类等。
- ❑ Zhongwen Development Tool(ZDT): <http://zdt.sourceforge.net/>, 学习中文的一个软件, 值得庆幸的是该软件是中国人开发的。如图 22.2 所示为该软件主界面的效果图。

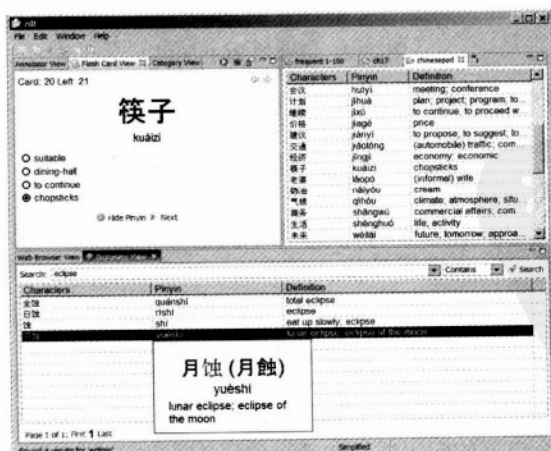


图 22.2 ZDT 软件的主界面

2. 商业的应用

- ❑ Actuate BIRT Report Designer: <http://www.actuate.com/birt>, BIRT 是 Eclipse 的一个报表开发引擎的插件, 该软件是 BIRT 报表的设计可视化客户端软件。使用该软件可轻松地创建所需的报表。
- ❑ Bay Breeze Software - SQL Edge: <http://www.baybreezesoft.com/>, 提供了可视化的执行 SQL 语句的工作环境, 能够显示环境数据库中表的关系图, 也能够浏览表中的数据。
- ❑ BSI CRM on Eclipse: <http://www.bsiag.com/joomla/index.php>, CRM 客户关系管理软件, 包括联系人管理、任务管理、项目管理和市场管理等。
- ❑ IBM Workplace Client Technology: <http://www-128.ibm.com/developerworks/workplace/products/clienttechnology/>, IBM 的客户管理软件。该软件的界面效果如图 22.3 所示。

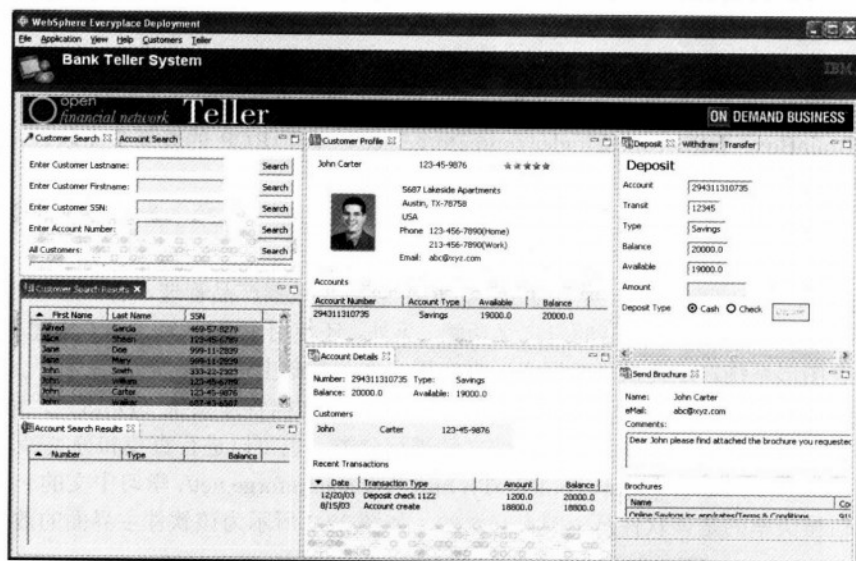



图 22.3 商业的 RCP 应用

 **注意:** 读者若想了解更多的 RCP 应用案例, 可以访问网站 <http://www.eclipse.org/community/rcp.php>。

从以上这些 RCP 的实现案例中可以看出, RCP 的应用非常广阔, 从系统的文件管理到 J2EE 的企业管理软件, 从可视化的报表工具到 3D 绘图, 都可以应用 RCP。只要是开发桌面的应用就都可以采用 RCP 来进行开发。

22.2 第一个 RCP 项目

了解了 RCP 有这么丰富的应用后, 读者一定想迫不及待地知道如何来开发 RCP 程序。那么本节将一步步地讲述如何来开发 RCP 程序。

22.2.1 创建插件项目

RCP 的开发基于 Eclipse 插件的开发, 所以创建 RCP 的项目也是从创建插件项目开始的。以下为创建一个 RCP 程序的步骤:

(1) 选择“文件”|“新建”|“项目”命令, 在弹出的“新建项目”对话框中选择“插件项目”, 单击“下一步”按钮, 弹出如图 22.4 所示的对话框。

(2) 输入项目名称为 MyRCP, 其他设置如图 22.4 所示。然后单击“下一步”按钮, 弹出如图 22.5 所示的输入插件项目详细信息的对话框。

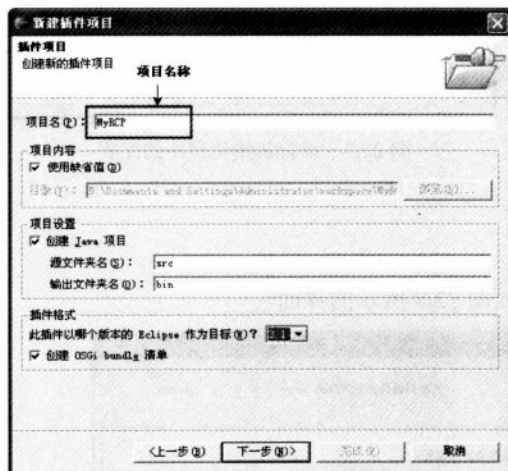


图 22.4 “新建插件项目”对话框

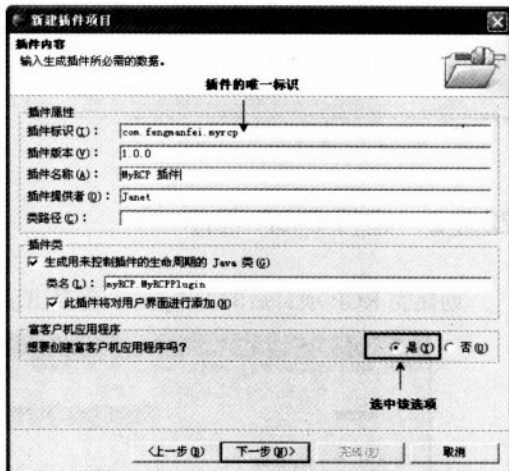


图 22.5 输入插件信息

(3) 在该对话框中可以设置一些插件的信息, 例如这里输入插件标识为“com.fengmanfei.myrpc”。为了保持唯一性, 不与其他插件混淆, 一般使用网址加名称的形式。但要注意, 一定要选中是否要创建富客户端应用程序选项。如果不选中, 创建的只是普通的 Eclipse 插件项目, 而不是 RCP 应用程序。按照如图 22.5 所示的设置完成后, 单击“下一步”按钮, 弹出如图 22.6 所示的对话框。

(4) 在该对话框中, 提供了创建 RCP 程序的几个常用的模板, 这里选择 Hello RCP 模板。这是最简单的 RCP 程序, 只有一个主界面。然后单击“下一步”按钮, 弹出如图 22.7 所示的对话框。

(5) 在该对话框中可以对 RCP 程序进行简单的配置, 如窗口显示的标题等, 这里使用向导默认的值。然后单击“完成”按钮, 这样一个 RCP 的项目就新建完成了。

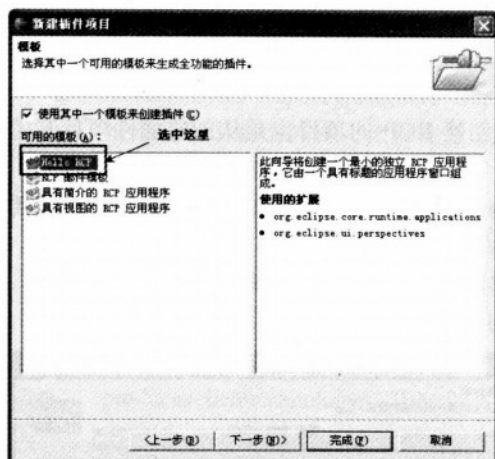


图 22.6 选择 RCP 程序的模板

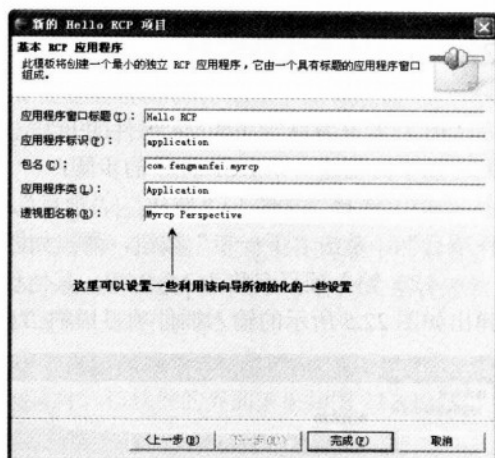


图 22.7 设置初始化程序的配置

22.2.2 运行 RCP 程序

创建完 RCP 项目后的 Eclipse 平台会出现如图 22.8 所示的界面。

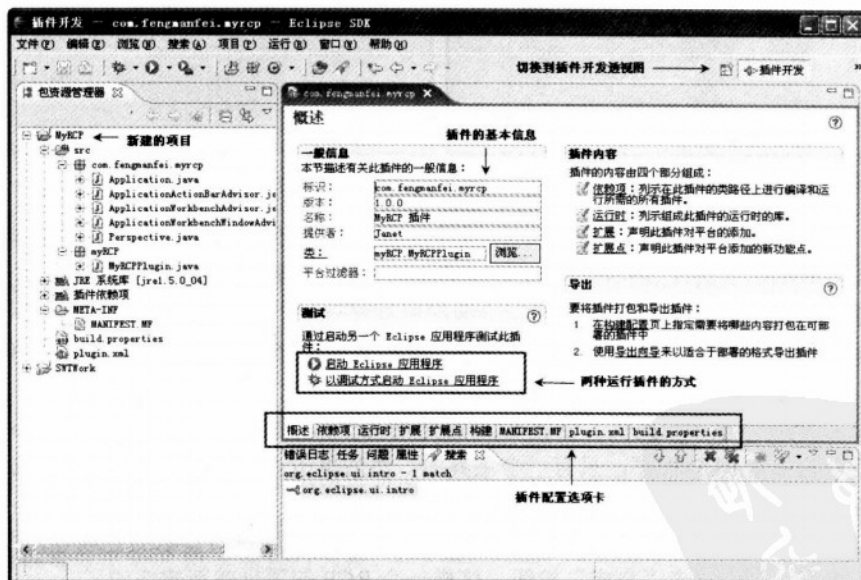


图 22.8 创建 RCP 项目后的界面

在左侧的项目文件中，会看到自动生成了一些 RCP 所需要的一些文件，这些都是利用创建向导自动生成的，是 RCP 程序运行所必需的一些文件。右侧是该项目的描述信息。可以单击右侧的标签来查看设置具体的插件配置参数。22.2.3 节将着重讲述这些参数和文件的

意义。要运行 RCP 程序, 有以下两种方法:

- ❑ 最简单的方法是在右侧“测试”信息框中, 单击“启动 Eclipse 应用程序”链接来运行。如果想要以调试方式运行, 则单击“以调试方式启动 Eclipse 应用程序”链接。
- ❑ 另外也可以用传统的方式运行 RCP 程序。在项目工程名称处右击, 在弹出的快捷菜单中选择“运行方式”|“Eclipse 应用程序”命令, 也可以运行 RCP 程序。

无论是哪种方式运行, 使用以上方法创建的 RCP 程序运行后的界面效果如图 22.9 所示。

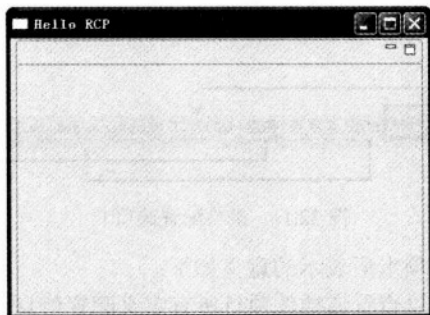


图 22.9 RCP 程序运行后的效果图

从图 22.9 中可以看出, 使用向导创建的 Hello RCP 程序很简单, 运行后只有一个主窗口界面, 这就是一个最简单的 RCP 程序。然后开发 RCP 的工作就是如何在该基础上进行开发, 按照需要来开发界面的效果。

用个比喻来说, 现在的这个 RCP 程序好比是一座空房子, 里面什么都没有, 接下来的开发的工作就是进行房子的装修, 按照设计来添置不同的家具。

22.2.3 插件的文件清单

在进行插件开发之前, 首先来看一下 22.2.2 节中利用向导建立的项目中各个文件所代表的意义。

- (1) 看一下自动生成的各个文件, 如图 22.10 所示。

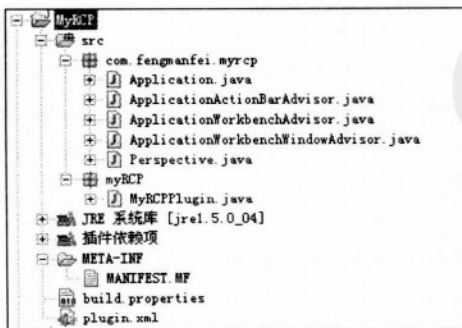


图 22.10 项目的文件结构

这些文件的具体说明如下：

- ❑ src 文件夹下为运行插件时的一些类文件。
- ❑ MANIFEST.MF 为插件清单文件，是插件与外界沟通的桥梁。
- ❑ build.properties 文件为构建 RCP 程序时所导入的类库设置。
- ❑ plugin.xml 是最重要的文件，该文件是插件的配置文件，集中管理插件内部的运行，在该文件中可以查找所有与该插件有关的信息。

(2) 看一下插件的配置信息。在如图 22.11 所示的选项卡中，可以单击切换不同的选项卡。



图 22.11 插件配置选项卡

如图 22.11 所示的各选项卡所表示的意义如下：

- ❑ “概述”选项卡可以查看该插件项目所有定义配置信息。
- ❑ “依赖项”、“运行时”选项卡是设置 MANIFEST.MF 文件的，在这两个选项卡所做的修改将同步应用于 MANIFEST.MF 文件。在 Eclipse 之前的版本中，没有 MANIFEST.MF 文件时，则是在 plugin.xml 文件中配置的。
- ❑ “扩展”选项卡是插件开发最重要的内容，各个插件都是基于扩展点的。在该选项卡中可以设置插件所使用的扩展点，并且在该选项卡中所做的配置将同步改变 plugin.xml 文件中的内容。
- ❑ “扩展点”选项卡用来设置该插件提供其他开发者的接口。如果想让其他开发人员对该插件进行开发，通常要提供这些开发人员一些接口，创建自定义的一些扩展点。
- ❑ “构建”选项卡可以可视化配置 build.properties 文件，“build.properties”选项卡可以查看该文件的源文件。

22.2.4 MANIFEST.MF 文件

MANIFEST.MF 文件是保存 OSGi 的 Bundle 文件。OSGi 开放服务网关协议（Open Services Gateway Initiative）是一个框架规范。OSGi 规范为网络服务定义了一个标准的、面向组件的计算环境，它最初的目的就是为各种嵌入式设备提供通用的软件运行平台，屏蔽设备操作系统与硬件区别的中间件平台。Eclipse OSGi 框架支持该规范。目前，OSGi 联盟发布的最新 OSGi 服务规范为 4.0。

通俗地讲，该文件也就是与其他平台的接口，通过该接口，外部平台可以使用该 Eclipse 的插件程序。如果读者有兴趣，可以访问 <http://www.osgi.org/>，以了解更多该规范的信息。

1. MANIFEST.MF 文件清单

下面具体来看一下 MANIFEST.MF 文件所配置的信息，如图 22.12 所示为 MANIFEST.MF 文件。

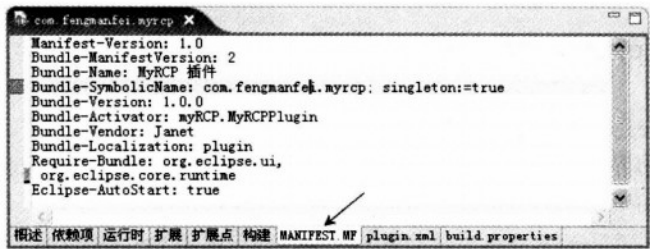


图 22.12 MANIFEST.MF 文件

其中，可以发现这些信息都是以 key 和 value 值来描述的。这些 key 值所表示的意义，可以参考 Eclipse 自带的参考帮助文档。以下是几个关键的 key 值所表示的意义。

- ❑ Bundle-Name: 插件的名称。
- ❑ Bundle-SymbolicName: 为插件的唯一标识。
- ❑ Bundle-Activator: 为主程序启动的类的全名。
- ❑ Require-Bundle: 系统编译和运行的依赖项，可以加入其他所必需的插件。

2. 添加依赖项

在编译和运行程序时，可能需要使用其他插件类库，这时可以在“依赖项”选项卡中单击“添加”按钮，然后选择运行时所必需的插件或者是某一个插件的类包。如图 22.13 所示为导入了 Ant 构建包后的界面。



图 22.13 导入依赖项

这样添加依赖项后，MANIFEST.MF 文件会加入以下的代码：

```
Require-Bundle: org.eclipse.ui,  
org.eclipse.core.runtime,  
org.eclipse.ant.core
```

此时可以看到，刚才导入的 org.eclipse.ant.core 自动添加到了 Require-Bundle 的值中。另外需要注意的是，这些类包加载是按照顺序进行的，所以可以使用“向上”和“向下”按钮进行调整。当然，运行时加载的插件只要能满足系统运行即可，越少越好，因为太多了会影响插件运行的加载速度。

22.2.5 build.properties 文件

build.properties 是保存构建、打包和导出插件所需的所有信息的文件。如图 22.14 所示为配置构建文件的界面。在该界面上的设置将会保存到 build.properties 文件中。例如,此时该文件中的代码如下:

```
source.. = src/  
output.. = bin/  
bin.includes = META-INF/\  
               .\  
               bin/  
jars.compile.order = .
```

其中,source 表示源文件的路径,output 表示编译后的.class 文件所在的路径,bin.includes 表示打包后包中所包含的文件,jars.compile.order 为编译的顺序。

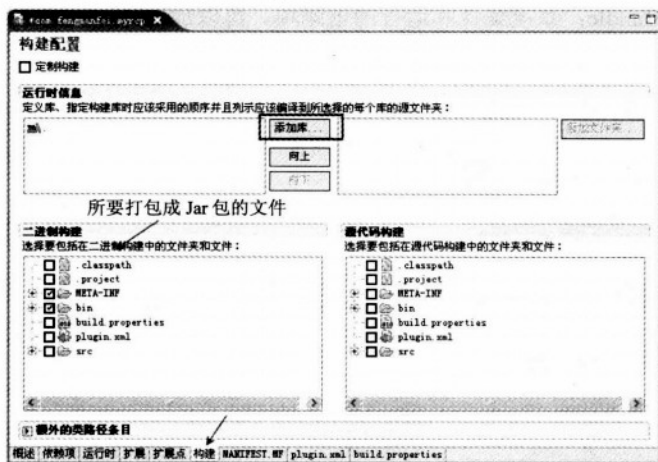


图 22.14 构建界面

22.2.6 plugin.xml 文件

plugin.xml 文件是插件开发中最重要的文件,包含声明插件的扩展和扩展点配置。如图 22.15 所示为该文件的源代码界面。

下面仔细分析一下各个扩展点的意义。每个扩展点都是一个<extension>元素。下面以第一个扩展点为例,说明各属性的意义。

- ❑ id="application", 表示该扩展点的标识。
- ❑ point="org.eclipse.core.runtime.applications", 表示扩展点的类型。这里表示该扩展点是系统扩展点,例如另一个扩展点类型“org.eclipse.ui.perspectives”为透视图扩展点。
- ❑ <run>子项中定义了系统启动的 class 类,为 com.fengmanfei.myrpc.Application,可

以在源文件下找到这个.java 源文件。

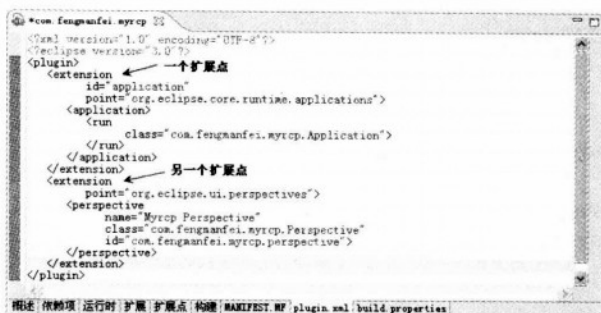


图 22.15 plugin.xml 文件

对于不同的扩展点有不同的配置元素，读者可以参考 Eclipse 的帮助文档。另外，读者无须手动输入这些配置文件，使用 Eclipse 自带的创建扩展点向导就可以轻松地完成创建扩展点的工作。

22.3 RCP 运行的基本原理

了解了这些配置文件后，下面仔细讲解一下整个 RCP 运行的过程。在 22.2 节创建的项目中，已经自动生成了 MyRCPPlugin、Application、ApplicationActionBarAdvisor、ApplicationWorkbenchAdvisor 和 Perspective 类。这些类是怎样联系起来的呢？就是以下所要学习的内容。

22.3.1 插件类 MyRCPPlugin

系统运行后首先在 MANIFEST.MF 文件中找到 Bundle-Activator 所对应的插件类，本项目中是 myRCP.MyRCPPlugin 类，表示是在 myRCP 包下的 MyRCPPlugin 类和该类的具体代码如下：

MyRCPPlugin.java

```
package myRCP;

import org.eclipse.ui.plugin.*;
import org.eclipse.jface.resource.ImageDescriptor;
import org.osgi.framework.BundleContext;
/**
 * 该插件类继承自 AbstractUIPlugin 类
 */
public class MyRCPPlugin extends AbstractUIPlugin {
    //插件类的对象，为静态对象
    private static MyRCPPlugin plugin;
    /**
```

```

    * 构造方法
    */
    public MyRCPPugin() {
        plugin = this;
    }
    /**
     * 装载 Bundle 文件并启动插件
     */
    public void start(BundleContext context) throws Exception {
        super.start(context);
    }
    /**
     * 插件运行结束后执行该方法
     */
    public void stop(BundleContext context) throws Exception {
        super.stop(context);
        plugin = null;
    }
    /**
     * 返回该插件的对象
     */
    public static MyRCPPugin getDefault() {
        return plugin;
    }
    /**
     * 返回相对于插件包文件下的相对路径的图片文件
     *
     * @param path the path
     * @return the image descriptor
     */
    public static ImageDescriptor getImageDescriptor(String path) {
        return AbstractUIPlugin.imageDescriptorFromPlugin("com.fengmanfei.myrpc", path);
    }
}

```

该类主要创建整个插件的对象，是插件的全局对象。可以通过该类的静态方法获得插件对象的引用，然后获得插件的各种信息，包括插件所对应的 Bundle 文件信息、log 日志对象和工作台对象等。例如，以下代码可以获得该插件在 MANIFEST.MF 定义的标识符。

```
MyRCPPugin.getDefault().getBundle().getSymbolicName();
```

通过 MyRCPPugin.getDefault() 还能获得插件的其他对象。各种方法请读者参阅 Eclipse API 帮助文档。

22.3.2 应用程序类 Application

创建了插件类后，如何来创建主窗口呢？Eclipse 运行的程序都是通过扩展点来配置的，初始化的界面程序也不例外，也是在配置的插件文件中配置好的。创建了插件类后，程序

会自动查找 plugin.xml 文件，是否有对应的应用程序扩展点 org.eclipse.core.runtime.applications。如果有，再找到该运行程序所对应的 class，然后创建对象。本例中为 com.fengmanfei.myrpc.Application，表示为 com.fengmanfei.myrpc 包下的 Application 类。具体这个类的代码如下：

Application.java

```
package com.fengmanfei.myrpc;

import org.eclipse.core.runtime.IPlatformRunnable;
import org.eclipse.swt.widgets.Display;
import org.eclipse.ui.PlatformUI;

/**
 * 该类实现了 IPlatformRunnable 接口
 */
public class Application implements IPlatformRunnable {
    /* (non-Javadoc)
     * @see org.eclipse.core.runtime.IPlatformRunnable#run(java.lang.Object)
     */
    public Object run(Object args) throws Exception {
        Display display = PlatformUI.createDisplay();
        try {
            int returnCode = PlatformUI.createAndRunWorkbench(display, new Application
                WorkbenchAdvisor());
            if (returnCode == PlatformUI.RETURN_RESTART) {
                return IPlatformRunnable.EXIT_RESTART;
            }
            return IPlatformRunnable.EXIT_OK;
        } finally {
            display.dispose();
        }
    }
}
```

在以上的代码中可以看到，首先创建了 Display 对象，然后调用 PlatformUI 的静态方法 createAndRunWorkbench 创建了 ApplicationWorkbenchAdvisor 对象。Application Workbench Advisor 对象表示的是工作台对象，用于创建工作台对象。

该类的代码如下：

ApplicationWorkbenchAdvisor.java

```
package com.fengmanfei.myrpc;

import org.eclipse.ui.application.IWorkbenchWindowConfigurer;
import org.eclipse.ui.application.WorkbenchAdvisor;
import org.eclipse.ui.application.WorkbenchWindowAdvisor;
//继承自 WorkbenchAdvisor 类，该类为抽象类
public class ApplicationWorkbenchAdvisor extends WorkbenchAdvisor {
```



```
//定义默认的透视图 id 字符串常量
    private static final String PERSPECTIVE_ID = "com.fengmanfei.myrpc.perspective";
//覆盖父类的方法
    public WorkbenchWindowAdvisor createWorkbenchWindowAdvisor(IWorkbenchWindow
    Configurer configurer) {
        return new ApplicationWorkbenchWindowAdvisor(configurer);
    }
//实现的抽象方法
    public String getInitialWindowPerspectiveId() {
        return PERSPECTIVE_ID;
    }
}
```

该类继承自 `WorkbenchAdvisor` 类。由于 RCP 程序启动后必须指定一个默认的透视图，所以要实现 `getInitialWindowPerspectiveId` 方法，则该方法返回一个透视图的 ID。例如，本例中使用的是 "com.fengmanfei.myrpc.perspective" 字符，表示首先在 `plugin.xml` 文件中找到 id 为该字符的 `<perspective>` 元素，然后根据这个 id 找到对应的 class 类。本例中对应的透视图类是 `Perspective`。

覆盖父类中的方法 `createWorkbenchWindowAdvisor` 中将创建一个工作台的窗口类，这里创建的是 `ApplicationWorkbenchWindowAdvisor` 对象。

到目前为止，以上所讲的这些类基本上不需要再改动，一个 RCP 程序总是按照这几个类所定义的模式创建。

22.3.3 工作台窗口类

下面看一下工作台的窗口类，通过工作台对象创建了工作台窗口对象 `ApplicationWorkbenchWindowAdvisor`。该类的代码如下：

ApplicationWorkbenchWindowAdvisor.java

```
package com.fengmanfei.myrpc;

import org.eclipse.swt.graphics.Point;
import org.eclipse.ui.application.ActionBarAdvisor;
import org.eclipse.ui.application.IActionBarConfigurer;
import org.eclipse.ui.application.IWorkbenchWindowConfigurer;
import org.eclipse.ui.application.WorkbenchWindowAdvisor;

public class ApplicationWorkbenchWindowAdvisor extends WorkbenchWindowAdvisor {
    public ApplicationWorkbenchWindowAdvisor(IWorkbenchWindowConfigurer configurer) {
        super(configurer);
    }
//覆盖父类中的方法，创建操作集 Action，也就是菜单栏、工具栏等对应的 Action
    public ActionBarAdvisor createActionBarAdvisor(IActionBarConfigurer configurer) {
        return new ApplicationActionBarAdvisor(configurer);
    }
}
```

//在打开窗口之前对窗口进行设置

```
public void preWindowOpen() {
    IWorkbenchWindowConfigurer configurer = getWindowConfigurer();
    configurer.setInitialSize(new Point(400, 300)); //设置窗口的初始大小
    configurer.setShowCoolBar(false); //设置是否显示工具栏
    configurer.setShowStatusLine(false); //设置是否显示状态栏
    configurer.setTitle("Hello RCP"); //设置窗口显示的标题
}
```

该类创建时首先对工作台进行设置。是在 `preWindowOpen` 方法中实现的。该方法会在窗口打开之前调用。另外，也可以通过覆盖父类中的方法来进行窗口打开之前的配置工作和窗口关闭的善后处理工作。以下列举了可覆盖的方法。

- ❑ `preWindowOpen`: 在打开窗口前调用该方法。
- ❑ `postWindowRestore`: 窗口还原后调用该方法。
- ❑ `postWindowCreate`: 窗口创建后调用该方法。
- ❑ `openIntro`: 打开介绍窗口时调用该方法。
- ❑ `postWindowOpen`: 当窗口打开后，调用该方法。
- ❑ `preWindowShellClose`: 窗口关闭前调用该方法。

另外，还可以覆盖以下 3 种方法。

- ❑ `createActionBarAdvisor`: 创建菜单和工具栏。
- ❑ `createEmptyWindowContents`: 当窗口中没有页面时，可在该方法中设置自定义的控件。
- ❑ `createWindowContents`: 可创建自定义的工作环境。一般不需要覆盖。

22.3.4 操作类

创建窗口时，会创建菜单栏和工具栏，此时创建的是 `ApplicationActionBarAdvisor` 对象。该类的代码如下：

ApplicationActionBarAdvisor.java

```
package com.fengmanfei.myrpc;

import org.eclipse.jface.action.IMenuManager;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.application.ActionBarAdvisor;
import org.eclipse.ui.application.IActionBarConfigurer;

public class ApplicationActionBarAdvisor extends ActionBarAdvisor {
    public ApplicationActionBarAdvisor(IActionBarConfigurer configurer) {
        super(configurer);
    }
    protected void makeActions(IWorkbenchWindow window) {
    }
    protected void fillMenuBar(IMenuManager menuBar) {
```

```
}  
}
```

在该类中，可以添加各种操作项来创建菜单栏、工具栏和状态栏等。在上面的代码中，还没有进行任何改动，也没有添加各种操作。后文将详细讲述这方面的内容。

22.3.5 透视图类

最后，在创建工作台对象时，要指定一个默认的透视图对象。本例中透视图类为 `Perspective`。该类的代码如下：

Perspective.java

```
package com.fengmanfei.myrpc;  
  
import org.eclipse.ui.IPageLayout;  
import org.eclipse.ui.IPerspectiveFactory;  
public class Perspective implements IPerspectiveFactory {  
    public void createInitialLayout(IPageLayout layout) {  
    }  
}
```

要实现一个透视图类，就要实现 `IPerspectiveFactory` 接口。这里先空实现它。具体的使用方法将在后文分别讲述。

综上所述，现在已经对一个最简单的 RCP 程序的原理弄清楚了。这部分内容比较抽象，不太容易理解，如果读者此时还有疑惑也没有关系。在后文中，将以实例来讲述，读者就会有比较直接的认识了。

22.4 创建扩展的基本方法

从前面内容可以了解到，开发插件最重要的是如何配置好扩展。幸运的是，Eclipse 已经提供了很好的配置扩展的工具，不用输入代码去配置，而是通过 UI 的界面就可以完成。下面详细讲述如何创建扩展的方法。

22.4.1 使用扩展向导创建

Eclipse 内置了一些常用的扩展点扩展的模板。使用模板可以快速地创建扩展。下面来看一下如何使用向导来创建。步骤如下：

(1) 在“扩展”选项卡中单击“添加”按钮，如图 22.16 所示。

(2) 在弹出的“新建扩展”对话框中选择“扩展向导”选项卡，在右侧的列表框中显示了一些已有的扩展模板，这里选择第一项，创建操作的扩展模板，如图 22.17 所示。



图 22.16 “扩展”选项卡页面

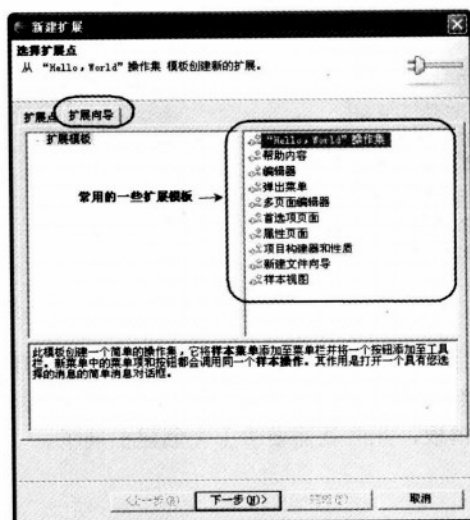


图 22.17 “扩展向导”选项卡

(3) 单击“下一步”按钮，按照选择的模板进行进一步的设置。这里使用默认值。然后单击“完成”按钮。这样，一个扩展自 `org.eclipse.ui.actionSets` 的扩展就完成了，如图 22.18 所示。



图 22.18 创建扩展后的界面

左侧列表中显示的是所有已扩展的元素。当单击左侧的一个扩展时，右侧会出现该扩

展元素的详细信息，然后可以在这里进行扩展的配置。当配置完成后，plugin.xml 文件中将会自动生成相应的源代码。按照以上步骤所创建扩展后的配置文件如下：

```
<extension
    point="org.eclipse.ui.actionSets">
    <actionSet
        id="com.fengmanfei.myrpc.actionSet"
        label="样本操作集"
        visible="true">
        <menu
            id="sampleMenu"
            label="样本菜单(&M)">
            <separator name="sampleGroup"/>
        </menu>
        <action
            class="com.fengmanfei.myrpc.actions.SampleAction"
            icon="icons/sample.gif"
            id="com.fengmanfei.myrpc.actions.SampleAction"
            label="样本操作(&S)"
            menubarPath="sampleMenu/sampleGroup"
            toolbarPath="sampleGroup"
            tooltip="Hello, Eclipse world"/>
        </actionSet>
    </extension>
```

这样就使用向导创建了一个操作集的扩展，并且相应地自动创建了该扩展所使用的一些类。很简单吧！不过虽然使用向导来创建很方便，但 Eclipse 提供的模板是有限的。若所要创建的扩展没有提供的模板，此时就需要手工去创建扩展了。

22.4.2 手工创建

手工创建虽然不如模板创建那样快捷和方便，但这也是很基础的工作。若要进行更高层次的开发，需要掌握手工创建的方法。手工创建扩展步骤如下：

(1) 与向导创建扩展的第一步相同，首先在“扩展”选项卡中单击“添加”按钮，然后在弹出的“新建扩展”对话框中选择“扩展点”选项卡，如图 22.19 所示。

(2) 如图 22.19 所示的列表中显示了所有可用的扩展点。若选中下方的“只显示必需插件中的扩展点”复选框，列表中则显示的是必要的扩展点。取消选中该复选框则会显示更多的扩展点。这里选中 org.eclipse.ui.views 扩展点，单击“完成”按钮。

(3) 这样就创建了简单的扩展点。在“扩展”选项卡界面上选中刚才创建的扩展，单击鼠标右键，弹出如图 22.20 所示的快捷菜单。

(4) 选择“新建”命令后将弹出子菜单，子菜单中可以创建的元素为当前扩展点可用的元素。可创建的元素也会根据所选择的扩展点而异。例如，本例是在 org.eclipse.ui.views 中新建的，可以新建的子元素为 category、view 和 stickyView。这里选择 view 即创建了一个 view 元素。创建完后，plugin.xml 文件中自动增加了以下扩展点配置信息：

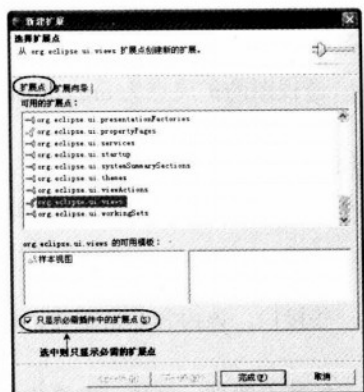


图 22.19 “扩展点”选项卡



图 22.20 创建 view 元素

```
<extension
    point="org.eclipse.ui.views">
    <view
        class="myRCP.ViewPart1"
        id="MyRCP.view1"
        name="MyRCP.view1"/>
</extension>
```

当然，手工配置的方式只是自动添加了配置文件，而扩展点所对应的类并没有自动创建，这些类还需要手工去创建。

最后需要注意的是，plugin.xml 配置信息都可以通过界面的操作方式来进行。所以下文中创建扩展时只列出最后配置的源代码部分，而不详细介绍配置的步骤。

22.4.3 获取扩展点的帮助

有这么多的扩展点，那么如何知道每个扩展点的意义及如何使用呢？Eclipse 为配置扩展也提供了很便捷查看帮助的方式。

(1) 在每个扩展点详细信息的下方有两个工具链接，如图 22.21 所示。

(2) 单击下方的“打开扩展点描述”链接，将会弹出该扩展点所对应的扩展点描述视图，如图 22.22 所示。

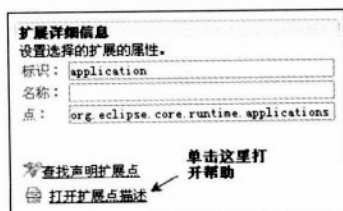


图 22.21 “打开扩展点描述”链接

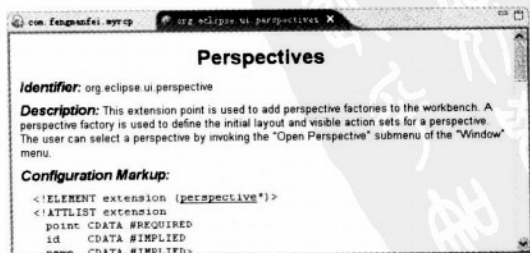


图 22.22 “扩展点的描述”视图

在该视图中就可以查看对应扩展点的详细信息了。在扩展点的描述信息中一般会有以下内容：

- ❑ **Identifier:** 扩展点的标识, 也就是扩展点的类型。例如, 透视图的扩展点为 `org.eclipse.ui.perspective`。
- ❑ **Description:** 扩展点的描述信息, 包括该扩展点的用途等。
- ❑ **Configuration Markup:** xml 标记的 Schema, 也就是说哪些元素是必需的, 哪些元素是可选的, 以及每个元素所表示的意义。
- ❑ **Examples:** 该扩展点配置的一个具体示例。
- ❑ **API Information:** 该扩展点所使用的类需要实现的一些接口。该内容非常重要, 需要仔细阅读。
- ❑ **Supplied Implementation:** Eclipse 平台中已经实现该扩展点的类。例如, 在操作扩展点, Eclipse 已经实现了一些“打开文件”、“退出”系统等常用的操作扩展, 在开发时可以直接使用这些扩展。

当然, 也可以通过 Eclipse 帮助文档来查看这些扩展点的详细信息。在帮助页面中选择“平台插件开发者指南”|“参考”|“扩展点参考”命令即可。

22.5 本章小结

本章主要介绍了 RCP 的应用。RCP 对于使用 SWT/JFace 开发桌面应用是一个很好的框架。RCP 的出现, 既利用了 Eclipse 插件开发的优势, 又脱离了对 Eclipse 的依赖, 有很广阔的应用前景。开发 RCP 程序最重要的就是扩展点的知识, 希望读者仔细阅读一下帮助文档中这部分内容, 为第 23 章进行实战开发打好基础。



第 23 章 RCP 开发

本章将详细讲述 SWT 中如何在 Eclipse 插件开发的框架中编写程序，包括如何创建菜单、工具栏，如何创建视图、如何创建编辑器、如何创建首选项、如何添加帮助文档、如何创建欢迎界面、如何创建在线更新等应用程序常用的功能。

23.1 扩展操作集 (Action Set)

操作集 (Action Set) 是操作的集合，在 JFace 中，已经学习了使用 Action 对象来创建菜单、工具栏和上下文菜单。Eclipse 插件开发下也使用了这种模式，创建一个 Action，可以将该 Action 应用到菜单项、工具栏和上下文菜单中。

Eclipse 插件开发操作集有两种方式：一种是扩展的方式，在 plugin.xml 文件中配置操作集，然后 Eclipse 平台会相应地创建菜单；另一种是编程的方式，这与之之前 JFace 中学习使用创建 Action 的方法相同。下面就分别介绍这两种创建操作集的具体方法。

23.1.1 操作集扩展点

在 22.4 节中讲述如何使用向导创建扩展时，就是以创建操作扩展为示例。下面就修改该扩展。

(1) 打开扩展的配置页面进行该扩展的配置，配置完成后，plugin.xml 文件中该扩展的具体代码如下：

```
<extension
    point="org.eclipse.ui.actionSets">
    <actionSet
        id="com.fengmanfei.myrpc.actionSet"
        label="文件"
        visible="true">
        <menu
            id="fileMenu"
            label="文件菜单(&F)">
            <separator name="fileGroup"/>
        </menu>
        <action
            class="com.fengmanfei.myrpc.actions.SampleAction"
            icon="icons/sample.gif"
            id="com.fengmanfei.myrpc.actions.SampleAction"
            label="新建(&N)"
```

```
menubarPath="fileMenu/fileGroup"
toolbarPath="fileGroup"
tooltip="Hello, Eclipse world"/>
</actionSet>
</extension>
```

(2) 打开 `ApplicationWorkbenchWindowAdvisor.java` 的源文件, 找到如下所示的一行:

```
configurer.setShowCoolBar(false);
```

然后修改成

```
configurer.setShowCoolBar(true);
```

这样, 就可以设置工作台显示工具栏了。

(3) 在配置文件中, 可以看到对应的操作的 class 值为 “com.fengmanfei.myrpc.actions.SampleAction”, 也就是说, 该操作具体执行的结果是由这个类负责的, 该类必须实现 `IWorkbenchWindowActionDelegate` 接口。如下为该类的具体代码:

SampleAction.java

```
package com.fengmanfei.myrpc.actions;

import org.eclipse.jface.action.IAction;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.IWorkbenchWindowActionDelegate;
import org.eclipse.jface.dialogs.MessageDialog;

public class SampleAction implements IWorkbenchWindowActionDelegate {
    private IWorkbenchWindow window;
    public SampleAction() {}
    //具体处理事件的方法, 单击该菜单项时调用
    public void run(IAction action) {
        MessageDialog.openInformation(
            window.getShell(),
            "MyRCP 插件",
            "Hello, Eclipse world");
    }
    public void selectionChanged(IAction action, ISelection selection) {}
    public void dispose() {}
    public void init(IWorkbenchWindow window) {
        this.window = window;
    }
}
```

当单击菜单项或是工具栏中该按钮时, 将调用 `run()` 方法, 这里单击菜单, 则弹出一个消息对话框。

(4) 最后重新运行程序, 运行后的界面效果如图 23.1 所示。

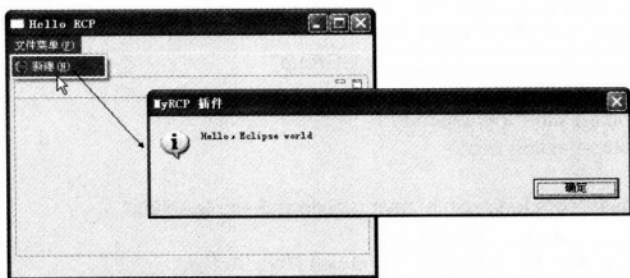


图 23.1 添加菜单后的界面效果

(5) 按照以上的步骤再多创建几个操作，这里为了演示最终的效果，将操作所对应的 Action 类都指向同一个类，而在实际的开发中通常是一个 Action 对应一个 class。配置完成后的 plugin.xml 文件中的代码如下：

```
<extension
    point="org.eclipse.ui.actionSets">
    <actionSet
        id="com.fengmanfei.myrpc.actionSet"
        label="文件"
        visible="true">
        <menu
            id="fileMenu"
            label="文件菜单(&F)">
            <separator name="fileGroup"/>
            <separator name="toggleGroup"/>
            <separator name="radioGroup"/>
        </menu>
        <action
            class="com.fengmanfei.myrpc.actions.SampleAction"
            icon="icons/sample.gif"
            id="com.fengmanfei.myrpc.actions.SampleAction"
            label="新建(&N)"
            menubarPath="fileMenu/fileGroup"
            toolbarPath="fileGroup"
            tooltip="Hello, Eclipse world"/>
        <action
            class="com.fengmanfei.myrpc.actions.SampleAction"
            icon="icons/sample.gif"
            id="com.fengmanfei.myrpc.actions.OneAction"
            label="Toggle One"
            menubarPath="fileMenu/toggleGroup"
            style="toggle"
            toolbarPath="toggleGroup"
            tooltip="aciton one"/>
        <action
            class="com.fengmanfei.myrpc.actions.SampleAction"
            icon="icons/sample.gif"
```



```

        id="com.fengmanfei.myrpc.actions.TwoAction"
        label="Toggle Two"
        menubarPath="fileMenu/toggleGroup"
        style="toggle"
        toolbarPath="toggleGroup"
        tooltip="action two"/>
<action
    class="com.fengmanfei.myrpc.actions.SampleAction"
    icon="icons/sample.gif"
    id="com.fengmanfei.myrpc.actions.RadioAction"
    label="Radio One"
    menubarPath="fileMenu/radioGroup"
    style="radio"
    tooltip="radio one"/>
<menu
    id="twoMenu"
    label="Two Menu"
    path="fileMenu">
    <separator name="editGroup"/>
</menu>
<action
    class="com.fengmanfei.myrpc.actions.SampleAction"
    icon="icons/sample.gif"
    id="com.fengmanfei.myrpc.actions.PushAction"
    label="Push"
    menubarPath="twoMenu/editGroup"
    style="push"
    toolbarPath="editGroup"
    tooltip="push"/>
</actionSet>
</extension>

```

按以上代码进行配置后，程序运行的效果如图 23.2 所示。

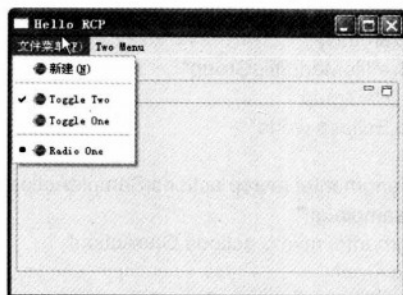


图 23.2 添加多个菜单项后的界面

到目前为止，还没有进行任何的编码就已经配置出这么多的菜单了，读者可以体会到使用插件开发带来的方便之处。下面就来具体分析一下配置菜单时所注意的问题。

- 要配置操作集，扩展点的类型是 `org.eclipse.ui.actionSets`。

- ❑ 一个<actionSet>元素表示具有一定结构的一组操作，可以为一个透视图指定一个操作集。
- ❑ 一个<menu>元素表示根目录的一个菜单，id 表示该菜单的标识，该值会被引用到显示菜单项的路径中，path 的值是标识菜单所在根目录的路径。例如，本例中 path="fileMenu"表示该菜单在 fileMenu 菜单后，若没有设定此值，默认值为 "additions"，则显示在最左方。
- ❑ 一个<separator>表示一组 Action，name 用于标识该组所在路径。
- ❑ 一个<action>表示一个具体的操作，该元素最重要的属性就是 id 和 class，id 用于区别于其他的操作，而 class 为该操作所对应的 Java 类，该类必须是实现以下两个接口中的任意一个接口的类，其中第二个当操作的 style 为 pulldown（下拉操作）时才使用。
- ❑ org.eclipse.ui.IworkbenchWindowActionDelegate。
- ❑ org.eclipse.ui.IworkbenchWindowPulldownDelegate。

一般来说，为了不与其他 Action 冲突，通常是 id 与 class 的值相同，这样就保持唯一了。

- ❑ <action>元素的 menubarPath 和 toolbarPath，是设置操作在菜单中的位置，都是以 "/" 来标识当前的位置的。例如，本例中 menubarPath="fileMenu/toggleGroup"表示的 id 为 fileMenu 的菜单下，separator 的 name 值为 toggleGroup 组中。而显示在工具栏中的路径，只需要使用<separator>路径即可。
- ❑ Action 中的<style>可以设置操作的样式，可以是 push、toggle、radio 或是 pulldown。
- ❑ 另外，还可以<action>作更加详细的设置，请读者阅读该扩展点的帮助文档，这里就不详细介绍了。

23.1.2 编写代码创建操作对象

除了通过配置扩展点的方式创建菜单和工具栏外，也可以通过编码的方式创建所需的菜单，这与 JFace 中创建菜单和工具栏的方式类似，下面就来具体看一下。

首先重新看一下 ApplicationActionBarAdvisor 这个类，可以在该类中使用编写代码的方式创建菜单和工具栏。现在修改该类的代码如下：

ApplicationActionBarAdvisor.java

```
package com.fengmanfei.myrpc;  
import org.eclipse.jface.action.Action;  
import org.eclipse.jface.action.ICoolBarManager;  
import org.eclipse.jface.action.IMenuManager;  
import org.eclipse.jface.action.IToolBarManager;  
import org.eclipse.jface.action.MenuManager;  
import org.eclipse.jface.action.ToolBarManager;  
import org.eclipse.ui.IWorkbenchWindow;  
import org.eclipse.ui.actions.ActionFactory;
```

```
import org.eclipse.ui.actions.ActionFactory.IWorkbenchAction;
import org.eclipse.ui.application.ActionBarAdvisor;
import org.eclipse.ui.application.IActionBarConfigurer;

public class ApplicationActionBarAdvisor extends ActionBarAdvisor {

    private NewAction newAction;//自定义的 action
    private IWorkbenchAction exitAction;//退出
    private IWorkbenchAction aboutAction;//关于

    public ApplicationActionBarAdvisor(IActionBarConfigurer configurer) {
        super(configurer);
    }

    protected void makeActions(IWorkbenchWindow window) {
        newAction = new NewAction();//创建 action 对象
        register(newAction);//父类方法，注册该操作
        exitAction = ActionFactory.QUIT.create(window);//Eclipse 内置的退出操作
        register(exitAction);
        aboutAction = ActionFactory.ABOUT.create(window);//内置的关于操作
        register(aboutAction);
    }
    //覆盖父类中的方法，创建菜单栏
    protected void fillMenuBar(IMenuManager menuBar) {
        MenuManager codeMenu = new MenuManager("CodeMenu(&C)");
        codeMenu.add(newAction);
        codeMenu.add(aboutAction);
        codeMenu.add(exitAction);
        menuBar.add(codeMenu);
    }
    //覆盖父类中的方法，创建工具栏
    protected void fillCoolBar(ICoolBarManager coolBar) {
        IToolBarManager toolbar = new ToolBarManager(coolBar.getStyle());
        toolbar.add(aboutAction);
        toolbar.add(newAction);
        coolBar.add(toolbar);
    }
}

//继承自 Action
class NewAction extends Action {
    NewAction() {
        super("new");
        this.setId("NewAction");
    }
    public void run() {
    }
}
```

修改代码后重新运行程序，此时界面的效果如图 23.3 所示。

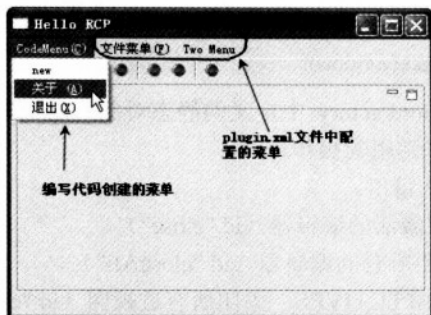


图 23.3 编写代码添加菜单后的效果图

从图中可以看出,不仅有使用编写代码的方式创建的菜单,还有通过 `plugin.xml` 文件配置的菜单,都显示在了菜单栏中。事实上,创建菜单栏对象时,首先加载配置文件中配置的操作,然后再加载通过代码编写的操作,最后统一装配显示出来。

23.1.3 编写代码创建操作的步骤

下面来具体分析一下代码的方式创建菜单的步骤:

1. 创建操作对象

通过覆盖父类的方法 `makeActions` 可以创建操作对象,这些操作对象都是实现了 `IAction` 接口的类的实例。创建操作对象有以下两种方法:

(1) 继承 `Action` 类。

代码中的 `NewAction` 对象就是这样创建的。这种用法与之前在 `JFace` 部分中使用的 `Action` 对象一样。但要注意,这里在继承 `Action` 类时,要设置该操作所对应的 `id`。其实这与 `plugin.xml` 文件中配置 `<action>` 元素时所指定的 `id` 一样,指定了 `id` 就相当于在程序运行时给该 `Action` 注册了一个标识,这样, `Eclipse` 就可以集中管理这些所有的操作对象了。当然,一般来说,通常将 `id` 设置为字符串常量的方式。

(2) 实现 `IWorkbenchAction` 接口。

事实上, `IWorkbenchAction` 接口继承自 `IAction` 类,只不过是在该接口中定义了 `dispose` 方法,以下就是该接口类的源代码:

```
public interface IWorkbenchAction extends IAction {
    public void dispose();
}
```

所以,也可以如以下方式定义操作类:

```
public class NewAction extends Action implements IWorkbenchAction {
    //中间代码省略
}
```

本例程序中,使用了 `Eclipse` 内置的一些操作,如退出、关闭视图、打开透视图,这些

操作一般都是一个 RCP 程序中很常用的。例如，创建一个退出按钮的对象所使用的代码是：

```
ActionFactory.ABOUT.create(window);
```

其中，ABOUT 为 ActionFactory 中定义的静态对象，除了本例所使用的两个内置的操作对象外，以下列举了一些常用的操作。

- ☐ ABOUT: 打开关于窗口。
- ☐ CLOSE: 关闭当前激活的编辑器 (id "close")。
- ☐ CLOSE_ALL: 关闭所有的编辑器 (id "closeAll")。
- ☐ CLOSE_ALL_PERSPECTIVES: 关闭所有透视图 (id "closeAllPerspectives")。
- ☐ CLOSE_ALL_SAVED: 关闭所有已保存的编辑器 (id "closeAllSaved")。
- ☐ CLOSE_PERSPECTIVE: 关闭当前透视图 (id "closePerspective")。
- ☐ COPY: 复制 (id "copy")。
- ☐ CUT: 剪切 (id "cut")。
- ☐ DELETE: 删除 (id "delete")。
- ☐ INTRO: 打开欢迎界面 (id "intro")。
- ☐ NEW_EDITOR: 新建一个编辑器 (id "newEditor")。
- ☐ OPEN_NEW_WINDOW: 在新窗口打开 (id "openNewWindow")。
- ☐ PASTE: 粘贴 (id "paste")。
- ☐ PREFERENCES: 打开首选项 (id "preferences")。
- ☐ PRINT: 打印 (id "print")。
- ☐ QUIT: 退出 (id "quit")。
- ☐ RESET_PERSPECTIVE: 重置当前的透视图 (id "resetPerspective")。

2. 注册操作对象

注册的方法很简单，调用父类中的 register 方法即可，例如以下代码：

```
register(aboutAction);
```

3. 创建菜单

创建菜单是通过覆盖父类的方法 fillMenuBar 来实现的，在该方法中可以创建 MenuManger 对象来创建菜单。这与之之前 JFace 中 MenuManager 的用法相同，但也可以将这些操作嵌入到 plugin.xml 文件所配置的菜单中。例如，将以上代码中创建 MenuManger 对象的一行改为：

```
MenuManager codeMenu = new MenuManager("CodeMenu(&C)","fileMenu");
```

则程序运行后的效果如图 23.4 所示。

从图中可以看出，编写代码的菜单嵌入到配置文件的菜单中，这是由于在创建 MenuManager 对象时指定了 id 值，也就是菜单所出现的路径，由于在 plugin.xml 文件中，有以下代码：

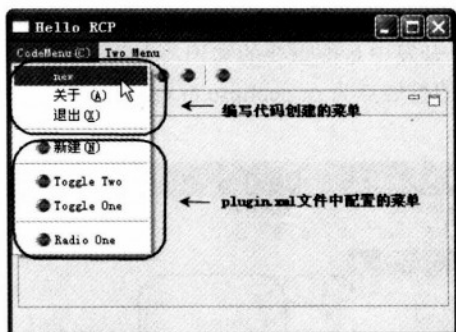


图 23.4 嵌入到配置的菜单中

```
<menu
  id="fileMenu"
  label="文件菜单(&F)">
  <separator name="fileGroup"/>
</menu>
```

这里指定的菜单的 id 为 “fileMenu”，也就相应地附加到该菜单中。

4. 创建工具栏

与创建菜单的方法类似，创建工具栏需要覆盖 fillCoolBar 方法，创建工具栏也可以通过 ToolManager 对象来实现，这里就不详细说明了。

综上所述，虽然是两种不同创建菜单的方式，但最后达到的效果都相同。每种方法各有利弊。通过配置的方法虽然简单，但不够灵活，尤其是不能创建系统内置的一些常用操作；而编写代码的方式虽然很传统，而且也不易集中管理，但提供了足够的灵活性。在实际的开发中一般都是两种方法同时使用，但要视具体情况而定。

23.1.4 其他与操作有关的扩展点

除了创建操作集（org.eclipse.ui.actionSets）扩展点外，还有一些与操作有关的扩展点。

- ❑ org.eclipse.ui.popupMenus: 创建上下文菜单的扩展点。
- ❑ org.eclipse.ui.editorActions: 为编辑器创建菜单的扩展点。
- ❑ org.eclipse.ui.viewActions: 为视图创建菜单的扩展点。
- ❑ org.eclipse.ui.commands: 为操作指定快捷键。

这些扩展的用法这里就不详细介绍了，请读者参阅相关的扩展点参考文档。

23.2 扩展视图

视图（View）是工作台页面内的可视组件。通常用来浏览信息的层次结构（如工作空间）、打开编辑器或显示活动编辑器的属性。例如在 Eclipse 工作区中，有问题视图、属性

视图等。

视图 (View) 在 RCP 开发中是很重要的知识, 大部分显示信息的主框架都是通过视图来显示的。用于创建视图的扩展点为 `org.eclipse.ui.views`。如图 23.5 所示为一个简单显示的视图界面效果。

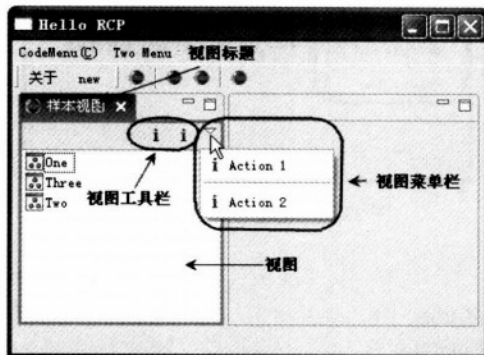


图 23.5 显示视图后的界面

23.2.1 视图扩展点

下面就来看一下如何实现图 23.5 所示效果的视图。首先找到扩展视图的扩展点 `org.eclipse.ui.views`, 配置扩展信息。配置好的 `plugin.xml` 文件如下:

```
<extension
    point="org.eclipse.ui.views">
    <category
        id=" myGroup "
        name="样本类别"/>
    <view
        allowMultiple="true"
        category=" myGroup "
        class="com.fengmanfei.myrpc.views.SampleView"
        icon="icons/sample.gif"
        id="com.fengmanfei.myrpc.views.SampleView"
        name="样本视图"/>
</extension>
```

其中, 主要元素的意义如下:

- ❑ 要配置视图, 扩展点的类型是 “`org.eclipse.ui.views`”。
- ❑ `<view>`元素中表示视图。其中 `class` 表示该视图所对应的类, 这里为 `SampleView` 类, 且该类必须实现 `org.eclipse.ui.IViewPart` 接口, 它的一个抽象实现为 `org.eclipse.ui.part.ViewPart`, 通常使用继承该类的方法。
- ❑ `<view>`元素中 `id` 表示该视图的唯一标识, 通过该标识可以在编程时获得该视图对象。例如在视图类中可以使用以下代码获得该视图对象:

```
IViewPart view = this.getSite().getPage().findView( "com.fengmanfei.myrpc.views.SampleView " );
```

其中，这里使用的就是在 plugin.xml 文件中配置的该视图的 id 值。

- ❑ <view>元素中，name 表示该视图显示的名称，icon 表示所显示的图标。
- ❑ <view>元素中，allowMultiple 表示该视图是否可以在同一个工作区创建多个视图，false 表示只允许显示一个该视图的实例。true 表示可以显示多个，如图 23.6 所示，为显示同一个界面上显示多个同一个视图时的界面效果。
- ❑ <category>表示可以为视图进行分类，id 为被视图分类的标识，name 为显示的名称，<view>元素中的<category>表示该视图所在的分类名称。如图 23.7 所示为 Eclipse 工作区视图的分类情况。

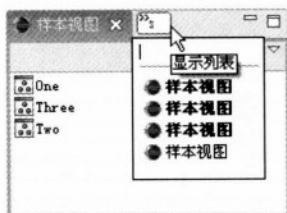


图 23.6 allowMultiple 为 true 时的界面效果图

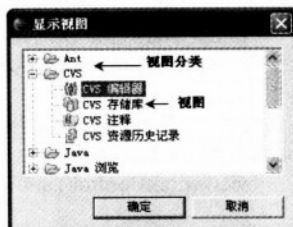


图 23.7 视图的分类

23.2.2 视图类

配置完视图的信息，下面来看一下该视图所对应的类如何实现，本例中该视图所对应的类为 com.fengmanfei.myrpc.views.SampleView，该类的具体代码如下：

SampleView.java

```
package com.fengmanfei.myrpc.views;

import org.eclipse.swt.widgets.Composite;
import org.eclipse.ui.part.*;
import org.eclipse.jface.viewers.*;
import org.eclipse.swt.graphics.Image;
import org.eclipse.jface.action.*;
import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.ui.*;
import org.eclipse.swt.widgets.Menu;
import org.eclipse.swt.SWT;

public class SampleView extends ViewPart {
    //该视图的 id，在 plugin.xml 文件中定义，通常使用字符串常量的方式
    public static final String ID = "com.fengmanfei.myrpc.views.SampleView";
    private TableViewer viewer;//视图中显示的表格对象
    //操作对象
    private Action action1;
    private Action action2;
```

```

//构造方法
public SampleView() {
}
//为父类中的抽象方法，创建视图中的各种控件
public void createPartControl(Composite parent) {
    //创建一个表格对象
    viewer = new TableViewer(parent, SWT.MULTI | SWT.H_SCROLL | SWT.V_SCROLL);
    viewer.setContentProvider(new ViewContentProvider());
    viewer.setLabelProvider(new ViewLabelProvider());
    viewer.setSorter(new NameSorter());
    viewer.setInput(getViewSite());
    hookDoubleClickAction();//添加表格双击事件
    makeActions();//创建操作对象
    hookContextMenu();//添加上下文菜单
    contributeToActionBars();//添加视图工具栏操作
}
//父类中的抽象方法，该视图获得焦点时，将焦点设置为表格
public void setFocus() {
    viewer.getControl().setFocus();
}
//表格的内容提供者
class ViewContentProvider implements IStructuredContentProvider {
    public void inputChanged(Viewer v, Object oldInput, Object newInput) {
    }
    public void dispose() {
    }
    public Object[] getElements(Object parent) {
        return new String[] { "One", "Two", "Three" };
    }
}
//表格的标签提供者
class ViewLabelProvider extends LabelProvider implements ITableLabelProvider {
    public String getColumnText(Object obj, int index) {
        return getText(obj);
    }
    public Image getColumnImage(Object obj, int index) {
        return getImage(obj);
    }
    public Image getImage(Object obj) {
        return PlatformUI.getWorkbench().
            getSharedImages().getImage(ISharedImages.IMG_OBJ_ELEMENT);
    }
}
class NameSorter extends ViewerSorter {
}
}

```

创建视图所对应的类时要注意以下几个问题：

- 创建视图类时，通常是继承抽象类 `ViewPart`，该类有两个抽象方法分别是

createPartControl()方法,负责创建视图中所显示的内容;setFocus()方法,为当前视图激活时,设置视图所在的焦点。

- ❑ 本例 createPartControl 中,创建了一个 TableView 的表格对象,用来显示视图中的数据。这里便涉及了之前学习的 SWT 和 JFace 创建 UI 界面的知识,所以在 RCP 中进行 UI 开发首先要打好 SWT 和 JFace 的基础。
- ❑ 有关获得图片的问题,在 Eclipse 平台运行时,图片资源是共享的,可以通过以下代码来获得 Eclipse 内置的一些图片:

```
PlatformUI.getWorkbench().getSharedImages().getImage(ISharedImages.IMG_OBJ_ELEMENT);
```

其中, ISharedImages.IMG_OBJ_ELEMENT 为 ISharedImages 类中封装的一些常量,读者可以参阅该类的帮助文档。

- ❑ 代码中涉及的一些与操作有关的方法,具体实现将在下文讲述。
- ❑ 最后,要想显示视图,还必须将该视图添加到默认的透视图中,所以 Perspective.java 类中的 createInitialLayout 方法的代码修改后如下:

Perspective.java

```
public class Perspective implements IPerspectiveFactory {  
    public void createInitialLayout(IPageLayout layout) {  
        String editorPart = layout.getEditorArea();  
        //为透视图添加一个视图  
        layout.addStandaloneView(SampleView.ID, //视图的 ID  
            true, //是否显示视图的标题  
            IPageLayout.LEFT, //放置在透视图的左侧位置  
            0.45f, //所占透视图的百分比为 45%  
            editorPart); //添加的相关区域  
    }  
}
```

其中, addStandaloneView 为在透视图中加入视图的方法,有关透视图的内容将会在 23.4 节详细讲述。

23.2.3 视图之间的交互

通常情况下,当双击一个视图中的内容时,另外一个视图中的内容要跟着改变,如图 23.8 所示,当单击左侧视图中的某一项时,下方的视图中的内容也随之改变。

要想实现这样的效果,按照以下所示的步骤进行:

(1) 创建另外一个视图,方法如 23.2.2 节创建视图的方法相同,该视图在 plugin.xml 文件的配置如下:

```
<view  
    allowMultiple="true"  
    class="com.fengmanfei.myrpc.views.AnotherView"
```



```
id="com.fengmanfei.myrpc.views.AnotherView"
name="另一个视图"/>
```

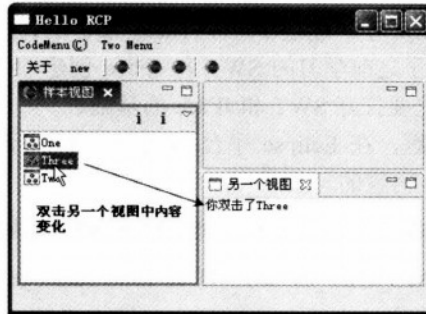


图 23.8 视图之间的事件交互

实现的视图类的代码如下：

AnotherView.java

```
package com.fengmanfei.myrpc.views;

import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Text;
import org.eclipse.ui.part.ViewPart;

public class AnotherView extends ViewPart {
    private Text text; // 文本框
    public static final String ID = "com.fengmanfei.myrpc.views.AnotherView";
    public AnotherView() {
        super();
    }
    public void createPartControl(Composite parent) {
        text = new Text(parent, SWT.NONE);
    }
    public void setFocus() {
        text.setFocus();
    }
    // 设置文本框的内容
    public void setContent(String content) {
        text.setText(content);
    }
}
```

(2) 为第一个视图中的 `ListViewer` 对象注册双击事件监听器，是由 `SampleView` 类中的 `hookDoubleClickAction` 方法实现的。以下是该方法中的具体代码：

```
private void hookDoubleClickAction() {
    // 注册双击事件监听器
    viewer.addDoubleClickListener(new IDoubleClickListener() {
```

```

//当双击时
public void doubleClick(DoubleClickEvent event) {
    //获得当前选中的一项
    ISelection selection = viewer.getSelection();
    Object obj = ((IStructuredSelection)selection).getFirstElement();
    //获得要显示内容的视图对象
    IViewPart viewPart = getSite().getPage().findView(AnotherView.ID);
    //确保找到所要使用的视图
    Assert.IsNotNull(viewPart);
    //强制类型转换为另一个视图对象
    AnotherView view = (AnotherView)viewPart;
    //设置该视图中显示的内容
    view.setContent( "你双击了" + (String)obj );
}
}
};
}

```

(3) 使视图显示在透视图的下方。修改 Perspective.java 的代码如下：

```

public void createInitialLayout(IPageLayout layout) {
    //之前的代码省略
    layout.addStandaloneView(AnotherView.ID,true,IPageLayout.BOTTOM,0.45f,editorPart);
}

```

这样程序重新分析一下该效果实现时要注意以下问题：

- 事件的代码处理主要是在 hookDoubleClickAction 方法中，关键是如何获得指定的视图对象，可以通过 getSite().getPage() 的 findView 方法来获得指定了 id 的视图对象。
- getSite().getPage() 方法返回的是 IWorkbenchPage 对象，该对象表示包含了当前工作区内的所有视图 (View) 和编辑器 (Editor)。除了可以查找当前的视图 findView 方法外，还有一些其他常用的方法，如下：
 - ◆ IViewPart findView(String viewId): 根据指定的视图 id，查找该视图对象。
 - ◆ IViewReference findViewReference(String viewId): 获得该视图的其他相关信息。
 - ◆ IEditorPart getActiveEditor(): 获得当前激活的编辑器对象。
 - ◆ IEditorPart findEditor(IEditorInput input): 查找指定的编辑器对象。
 - ◆ IEditorPart[] getEditors(): 获得所有的编辑器。
 - ◆ IEditorReference[] getEditorReferences(): 获得所有的编辑器相关信息。
 - ◆ IEditorPart[] getDirtyEditors(): 获得所有未保存的编辑器对象。
 - ◆ IPerspectiveDescriptor getPerspective(): 获得当前页面所属的透视图对象。
 - ◆ IViewReference[] getViewReferences(): 获得所有视图的相关信息。
 - ◆ IViewPart[] getViews(): 获得所有的视图对象。
 - ◆ openEditor(IEditorInput input, String editorId): 打开指定的编辑器。
 - ◆ openEditor(IEditorInput input, String editorId, boolean activate): 打开指定的编辑器，并可以设定编辑器的状态。
 - ◆ resetPerspective(): 重置透视图。

- ◆ saveAllEditors(boolean confirm): 保存所有的编辑器。
- ◆ saveEditor(IEditorPart editor, boolean confirm): 保存所有的编辑器, 并询问是否保存。
- ◆ savePerspective(): 保存透视图。
- ◆ savePerspectiveAs(IPerspectiveDescriptor perspective): 透视图另存为。
- ◆ setEditorAreaVisible(boolean showEditorArea): 设置编辑区是否显示。
- ◆ showView(String viewId): 显示指定的视图。
- ◆ showView(String viewId, String secondaryId, int mode): 显示指定的视图, 并可以指定视图显示的位置。
- ◆ closeEditor(IEditorPart editor, boolean save): 关闭指定的编辑器。
- ◆ closeAllEditors(boolean save): 关闭所有的编辑器, 并可以设置是否询问保存。

23.2.4 添加视图的工具栏

在视图的工具栏中, 一般可以放置一些常用的与该视图相关的操作, 通常也是使用 MenuManager 和 ToolBarManager 对象来创建的。本例中首先要创建这些 Action 对象, 是在 makeActions 方法中实现的。以下即为该方法的具体代码:

```
private void makeActions() {  
    action1 = new Action() {  
        public void run() {  
            showMessage("Action 1 executed");  
        }  
    };  
    action1.setText("Action 1");  
    action1.setToolTipText("Action 1 tooltip");  
    action1.setImageDescriptor(PlatformUI.getWorkbench().getSharedImages().  
        getImageDescriptor(ISharedImages.IMG_OBJS_INFO_TSK));  
  
    action2 = new Action() {  
        public void run() {  
            showMessage("Action 2 executed");  
        }  
    };  
    action2.setText("Action 2");  
    action2.setToolTipText("Action 2 tooltip");  
    action2.setImageDescriptor(PlatformUI.getWorkbench().getSharedImages().  
        getImageDescriptor(ISharedImages.IMG_OBJS_INFO_TSK));  
}
```

代码很简单, 只是初始化这些 Action 对象, 并将操作添加到工具栏中, 是在 contributeToActionBars 中实现的。该方法的代码如下:

```
private void contributeToActionBars() {  
    //获得视图的操作栏对象
```

```
IActionBars bars = getViewSite().getActionBars();
//添加下拉菜单
fillLocalPullDown(bars.getMenuManager());
//添加工具栏
fillLocalToolBar(bars.getToolBarManager());
}
//添加下拉菜单
private void fillLocalPullDown(IMenuManager manager) {
    manager.add(action1);
    manager.add(new Separator());
    manager.add(action2);
}
//添加工具栏
private void fillLocalToolBar(IToolBarManager manager) {
    manager.add(action1);
    manager.add(action2);
}
```

为视图添加操作栏主要是通过 `getViewSite().getActionBars()` 方法获得 `IActionBars` 对象, 然后使用该对象的 `getMenuManager` 方法获得菜单管理器对象, 然后再将 `Action` 添加到菜单中。同样, 工具栏是通过 `getToolBarManager` 方法获得工具栏管理器对象, 然后再添加的。

23.2.5 添加上下文菜单

通常也会为视图增加相应的上下文菜单。本例中是在 `hookContextMenu` 方法中实现的。该方法的代码如下:

```
private void hookContextMenu() {
    //创建菜单管理器对象
    MenuManager menuMgr = new MenuManager("#PopupMenu");
    menuMgr.add(action1);
    menuMgr.add(action2);
    menuMgr.add(new Separator(IWorkbenchActionConstants.MB_ADDITIONS));

    //为列表对象创建上下文菜单
    Menu menu = menuMgr.createContextMenu(viewer.getControl());
    //设置菜单
    viewer.getControl().setMenu(menu);
    //注册上下文菜单
    getSite().registerContextMenu(menuMgr, viewer);
}
```

很简单, 所需要的都是 `JFace` 中的知识。但是在实际的项目开发中, 通常上下文菜单会随着选中的内容不同而显示不同的菜单, 例如图 23.9 和图 23.10 所示, 当选中 `One` 时, `My Action` 菜单项不可用, 而当选中 `Three` 时, `My Action` 则为可用状态。那这种情况应该如何实现呢?



图 23.9 My Action 不可用状态

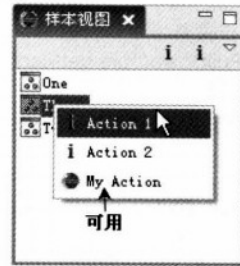


图 23.10 My Action 可用

要想实现这样的功能，需使用选择服务（SelectionService），首先来看一下代码，然后再来作一些详细的解释。

首先创建一个 MyAction 类。代码如下：

MyAction.java

```
package com.fengmanfei.myrpc.actions;

import org.eclipse.jface.action.Action;
import org.eclipse.jface.util.Assert;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.viewers.IStructuredSelection;
import org.eclipse.ui.ISelectionListener;
import org.eclipse.ui.IWorkbenchPart;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.actions.ActionFactory.IWorkbenchAction;
import myRCP.MyRCPPlugin;

public class MyAction extends Action implements ISelectionListener, IWorkbenchAction {
    private IWorkbenchWindow window;
    public final static String ID = "com.fengmanfei.myrpc.actions.MyAction";
    public MyAction(IWorkbenchWindow window) {
        super("Test Action");
        setId(ID);
        setText("My Action");
        setToolTipText("My Action");
        setImageDescriptor(MyRCPPlugin.getImageDescriptor("icons/sample.gif"));
        this.window = window;
        //注册选择服务监听器
        window.getSelectionService().addSelectionListener(this);
    }

    public void run() {
    }
    //接口 ISelectionListener 中的方法
    //当事件发生时调用该方法
    public void selectionChanged(IWorkbenchPart part, ISelection incoming) {
```



```

        if (incoming instanceof IStructuredSelection) {
            //获得事件发生的源所携带的对象
            IStructuredSelection selection = (IStructuredSelection) incoming;
            //强制转换
            String s = (String) selection.getFirstElement();
            Assert.isNotNull(s);
            //如果选中的是 Three, 则设置为可用, 否则设置为不可用
            if (s.equals("Three"))
                setEnabled(true);
            else
                setEnabled(false);
        }
    }
    //IWorkbenchAction 接口中的方法, 释放后取消注册
    public void dispose() {
        window.getSelectionService().removeSelectionListener(this);
    }
}

```

该 Action 代码与之前创建 Action 的代码有所不同, 实现了 ISelectionListener 接口, 目的是为了注册选择服务监听器, 关键是如下代码:

```

window.getSelectionService().addSelectionListener(this);

```

读者应该对 addXXXListener 的写法不陌生吧, 在注册事件监听器时也是这种用法。这段代码只是表示在选择服务中注册了这个监听器。但只有监听器是不够的, 还需要在服务中注册内容改变的提供者。所以, 在以上创建上下文菜单中要添加以下代码:

```

MenuManager menuMgr = new MenuManager("#PopupMenu");
//省略.....
menuMgr.add(new MyAction( getSite().getWorkbenchWindow()));
//注册内容改变的提供者
getSite().setSelectionProvider(viewer);

```

其中, setSelectionProvider(ISelectionProvider provider)方法将 viewer 对象在选择服务中注册, 原因是所有的 View 类, 包括 TableViewer、TreeViewer 和 ListViewer 都实现了 ISelectionProvider 接口。这样一旦将该 viewer 对象注册到了选择服务中, 该 viewer 对象发生改变时, 会通知到所有的监听器对象。

可以用图 23.11 来表示监听器和内容提供者与选择服务之间的关系。

一旦选择服务中的一个内容提供者变化, 会通知到所有的服务中注册的监听器对象。例如本例中的程序, 视图中的 ListView 对象就是注册到选择服务中的内容提供者, 当 ListView 选中的选项变化时, 会通知 MyAction 这个监听器, 执行 MyAction 中的 selectionChanged 方法, 此时根据选中的选项值来判断该 Action 的状态。

选择服务不仅可以应用到上下文菜单的判断中, 也可以用于视图与视图之间的联动控制。例如在 Eclipse 工作区中, 随着鼠标选中的选项不同, 属性视图中会出现不同信息, 如图 23.12 所示, 这些都是通过注册选择服务来实现的。

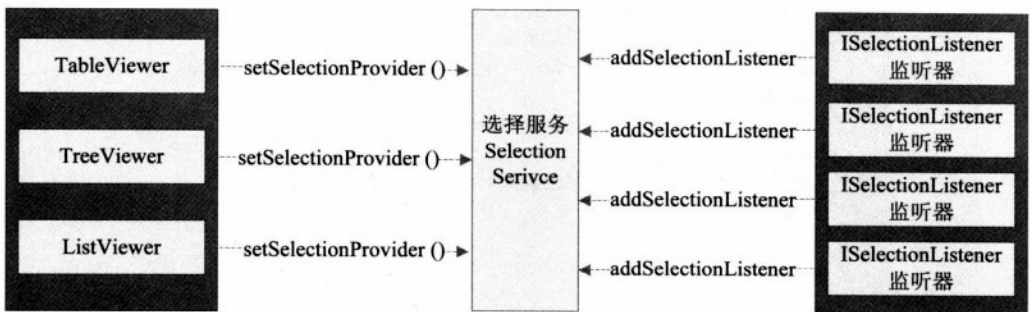


图 23.11 选择服务与内容提供器和监听器之间的关系



图 23.12 Eclipse 工作区的选择服务

23.3 扩展编辑器

编辑器与视图一样，是工作台页面内的可视组件。通常用来编辑文件（例如查源代码）或查看输入对象（例如打开的 `pugin.xml` 文件时的界面）。用于创建视图的扩展点为 `org.eclipse.ui.editors`。如图 23.13 所示即为一个打开的编辑器的页面效果。

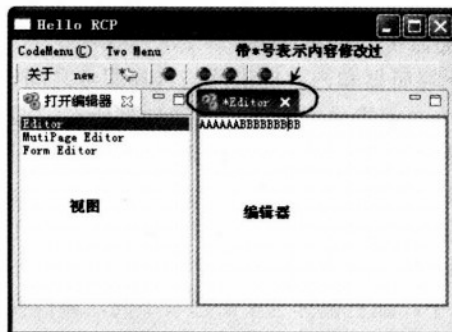


图 23.13 打开的编辑器效果

23.3.1 编辑器扩展点

下面就来看一下如何实现图 23.13 所示效果的编辑器。首先找到扩展编辑器的扩展点 `org.eclipse.ui.views`，配置扩展信息。配置好的 `plugin.xml` 文件如下：

```
<extension
    point="org.eclipse.ui.editors">
    <editor
        class="com.fengmanfei.myrpc.editors.JsEditor"
        contributorClass="com.fengmanfei.myrpc.editors.JsEditorContributor"
        default="false"
        icon="icons/samples.gif"
        id="com.fengmanfei.myrpc.editors.JsEditor"
        name="JsEditor"/>
    </extension>
```

其中，编辑器扩展点的各元素的说明如下：

- ❑ 编辑器的扩展点类型为 `org.eclipse.ui.editors`。
- ❑ 与视图一样，一个编辑器也对应一个 `class` 类，该类必须是实现了 `org.eclipse.ui.IEditorPart` 的类。
- ❑ `id` 是该编辑器对象的唯一标识，`name` 为编辑器显示的名称。
- ❑ `default`：表示是否使用默认的编辑器。
- ❑ `contributorClass` 是实现了 `org.eclipse.ui.IEditorActionBarContributor` 接口的类，Eclipse 中 `EditorActionBarContributor` 类已经实现了该接口，通常的做法是继承该类，然后覆盖父类中的方法实现的。
- ❑ 另外，除了以上所示的这几个元素属性外，还有一些其他的元素需要注意。如下：
 - ◆ `extensions`：打开该编辑器的所对应的文件的扩展名，例如“`htm, html`”。
 - ◆ `command` 和 `launcher`：`command` 为要运行以启动外部编辑器的命令。`Launcher` 为实现了 `org.eclipse.ui.IEditorLauncher` 的类。这样启动程序将打开外部编辑器。其中 `class`、`command` 和 `launcher` 都是打开编辑器的方式，属性是互斥的。
 - ◆ `filenames`：编辑器打开文件时可选的文件名。
 - ◆ `symbolicFontName`：编辑器字体的名称。
 - ◆ `matchingStrategy`：实现 `org.eclipse.ui.IEditorMatchingStrategy` 的类。这允许编辑器扩展提供一个策略，使编辑器的输入与指定的编辑器输入相匹配。

一般开发编辑器时通常会用到这些属性，但在 RCP 开发中，如果不是开发的 RCP 编辑器程序，一般是用不到这些属性的。

23.3.2 编辑器类

同视图一样，一个编辑器也对应一个 Java 类，本例中对应的类是 `com.fengmanfei.myrpc`。

editors.JsEditor。首先看一下该类实现的具体代码后，再详细地解释。该类的代码如下：

JsEditor.java

```
package com.fengmanfei.myrpc.editors;

import org.eclipse.core.runtime.IProgressMonitor;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.ModifyEvent;
import org.eclipse.swt.events.ModifyListener;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Text;
import org.eclipse.ui.IEditorInput;
import org.eclipse.ui.IEditorSite;
import org.eclipse.ui.PartInitException;
import org.eclipse.ui.part.EditorPart;

public class JsEditor extends EditorPart {
    //对应 plugin.xml 指定的 id
    public static final String ID = "com.fengmanfei.myrpc.editors.JsEditor";
    private Text text;
    //编辑器中的内容是否被修改的标志
    private boolean bDirty = false;
    public JsEditor() {
        super();
    }
    //初始化编辑器
    public void init(IEditorSite site, IEditorInput input)
        throws PartInitException {
        this.setSite(site); //设置 site
        this.setInput(input); //设置输入的 IEditorInput 对象
        this.setPartName(input.getName()); //设置编辑器上方显示的名称
    }
    //创建编辑器中的控件
    public void createPartControl(Composite parent) {
        text = new Text(parent, SWT.NONE);
        //当文本框修改时，设定内容被修改过
        text.addModifyListener(new ModifyListener() {
            public void modifyText(ModifyEvent e) {
                if (!isDirty()) { //如果未修改
                    setDirty(true); //设置修改
                    //更改编辑器的状态
                    firePropertyChange(IEditorPart.PROP_DIRTY);
                }
            }
        });
    }
    //编辑器关闭时，保存编辑器内容时所调用的方法
    public void doSave(IProgressMonitor monitor) {
```

```

//将保存状态显示在状态栏中
try {
    monitor.beginTask("保存文件...", 100);

    for (int i = 0; i < 10 && !monitor.isCanceled(); i++) {
        Thread.sleep(500);
        monitor.worked(10);
        double d = (i + 1) / 10d;
        monitor.subTask("已完成" + d * 100 + "%");// 显示任务状态
    }
    monitor.done();
    if (monitor.isCanceled())
        throw new InterruptedException("取消保存");
} catch (InterruptedException e) {
    ;
}

//另存为调用的方法
public void doSaveAs() {

}

//判断是否被修改过
public boolean isDirty() {
    return bDirty;
}

//是否允许保存
public boolean isSaveAsAllowed() {
    return true;
}

//设置焦点
public void setFocus() {
    text.setFocus();
}

//设置编辑器内容被修改过
public void setDirty(boolean b) {
    bDirty = b;
}
}

```

创建编辑器类时应该注意以下问题：

- ❑ 编辑器所对应的类为实现了 `IEditorPart` 接口的类，Eclipse 插件开发中已经实现了该接口的类如图 23.14 所示。通常可以继承这些类，本例中使用的是继承了 `EditorPart` 类，例如下文中讲述的多页编辑器则要继承自 `MultiPageEditorPart` 类，表单编辑器则要继承 `FormEditor` 类。
- ❑ 另外，在创建编辑器扩展的向导中，有一个 XML 编辑器的示例程序，该程序继承的则是 `TextEditor`。一般开发代码编辑器会继承该类，读者可以阅读相关的代码。
- ❑ 编辑器中显示的控件是 `createPartControl` 方法，本例中创建了一个很简单的文本框。

- 另外需要注意的是几个有关编辑器状态的方法，如下所示：
 - ◆ `isDirty()`：判断编辑器内容是否被修改过，如果被修改过，标签上会显示一个“*”号，如图 23.15 所示。



图 23.14 实现了 IEditorPart 接口的类

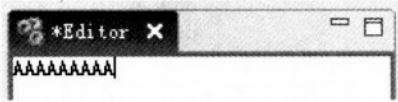


图 23.15 内容被修改后的编辑器的界面效果

本例中判断是否修改是根据一个布尔型的变量 `bDirty` 来判断的，当文本框的内容修改时，则将该值设置为 `true`。另外，当状态改变时要及时地更新界面的效果，此时要使用以下方法，将界面改成已修改的效果。

`firePropertyChange(IEditorPart.PROP_DIRTY)`

- ◆ `isSaveAsAllowed()`：是否允许全部保存。
- ◆ `doSaveAs()`：另存为时调用的方法。
- ◆ `doSave()`：保存时调用的方法。本例中，当保存编辑器时设置在状态栏中显示相应的进度，如图 23.16 所示。其中使用 `monitor` 对象就可以了，有关进度条的使用方法请读者参阅上文有关 JFace 的部分。



图 23.16 保存时显示保存进度效果

- ◆ `setFocus()`：设置编辑器激活时焦点所在的控件。

23.3.3 打开编辑器

与视图不同，视图不能直接显示到透视图的某一个区域，而是由一项操作所打开的。通常使用 `IWorkbenchPage` 的以下两种方法：

- `openEditor(IEditorInput input, String editorId)`：其中 `input` 对象为打开文件器时所指定输入到编辑器的内容，为了实现 `IEditorInput` 接口的类，`editorId` 则为编辑器的唯一标识。

- ❑ `openEditor(IEditorInput input, String editorId, boolean activate)`: 其中, `activate` 表示是否打开后并激活该编辑器。

本例中所创建的实现了接口的类为 `JsEditorInput` 类。该类的具体代码如下:

JsEditorInput.java

```
package com.fengmanfei.myrpc.editors;

import org.eclipse.jface.resource.ImageDescriptor;
import org.eclipse.ui.IEditorInput;
import org.eclipse.ui.IPersistableElement;
import myRCP.MyRCPPlugin;

public class JsEditorInput implements IEditorInput {
    //输入的字符
    private String input ;
    public JsEditorInput ( String input ){
        this.input = input ;
    }
    //是否将编辑器保存在最近访问的记录中
    public boolean exists() {
        return true;
    }
    //输入内容的图标
    public ImageDescriptor getImageDescriptor() {
        return MyRCPPlugin.getImageDescriptor("icon/sample.gif");
    }
    //输入信息的名称
    public String getName() {
        return input;
    }
    //是否可以持久化该编辑器
    public IPersistableElement getPersistable() {
        return null;
    }
    //设置编辑器标签中显示提示信息
    public String getToolTipText() {
        return input;
    }
    //返回与该输入相关的类的对象
    public Object getAdapter(Class adapter) {
        return null;
    }
}
```

这些方法都是 `IEditorInput` 接口中的方法。另外除了可以自己创建实现该接口的类外, Eclipse 中已经有一些实现了该接口的类, 可以直接使用或者是通过继承的方法。这些类如图 23.17 所示。

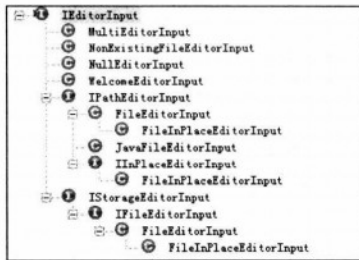


图 23.17 Eclipse 中已实现的 IEditorInput 接口的类

创建了打开编辑器输入内容的类，也有了对应的编辑器，就可以打开编辑器了。要想打开编辑器，还需要创建一个视图，视图中放置一个列表 List，然后单击列表中的一项来打开编辑器。按照前面创建视图的方法定义 plugin.xml 和对应类。

plugin.xml 文件中的配置代码如下：

```
<view
    class="com.fengmanfei.myrpc.views.OpenEditorView"
    icon="icons/samples.gif"
    id="com.fengmanfei.myrpc.views.OpenEditorView"
    name="打开编辑器"/>
```

对应的视图类如下：

OpenEditorView.java

```
package com.fengmanfei.myrpc.views;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.SelectionAdapter;
import org.eclipse.swt.events.SelectionEvent;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.List;
import org.eclipse.ui.IWorkbenchPage;
import org.eclipse.ui.PartInitException;
import org.eclipse.ui.part.ViewPart;
import com.fengmanfei.myrpc.editors.JsEditor;
import com.fengmanfei.myrpc.editors.JsEditorInput;

public class OpenEditorView extends ViewPart {
    public static final String ID = "com.fengmanfei.myrpc.views.OpenEditorView";
    public List list;
    public OpenEditorView() {
        super();
    }
    public void createPartControl(Composite parent) {
        list = new List(parent, SWT.NONE);
        list.add("Editor");
        list.add("MultiPage Editor");
        list.add("Form Editor");
    }
}
```

```

list.addSelectionListener(new SelectionAdapter() {
    public void widgetSelected(SelectionEvent e) {
        String select = list.getSelection()[0];
        //获得当前激活的 IWorkbenchPage 对象
        IWorkbenchPage page = getViewSite().getWorkbenchWindow().GetActive
Page();

        if (select.equals("Editor")) { //如果选中"Editor"项
            //创建输入的内容对象
            JsEditorInput editor = new JsEditorInput(select);
            try {
                //打开该编辑器
                page.openEditor(editor, JsEditor.ID);
            } catch (PartInitException ee) {
                ee.printStackTrace();
            }
        }
    }
});
}
public void setFocus() {
    list.setFocus();
}
}
}

```

这里，打开编辑器的代码主要在事件处理部分。

23.3.4 添加编辑器的菜单和工具栏

对于编辑器，也可以设置编辑器所对应的菜单、工具栏和上下文菜单。图 23.18 所示为编辑器激活时，该编辑器所使用的菜单和工具栏为可用状态。图 23.19 所示为编辑器未激活时，菜单和工具栏都处于不可用状态。

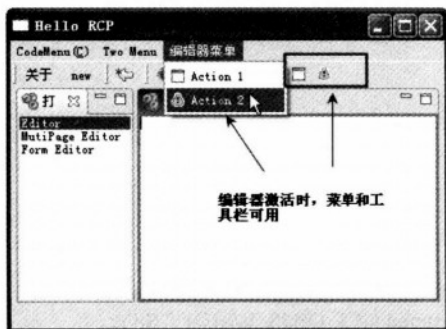


图 23.18 编辑器已激活时



图 23.19 编辑器未激活时

要想实现这样的效果，首先要配置该编辑器设置 contributorClass 属性，该属性对应一个 Java 类，本例中对应的是 com.fengmanfei.myrpc.editors.JsEditorContributor 类，该属性对

应的类必须是实现了 `IEditorActionBarContributor` 接口的类，而 `EditorActionBarContributor` 类已经实现了该类，所以 `JsEditorContributor` 类通过继承 `EditorActionBarContributor` 的方法来实现。该类具体的代码如下：

JsEditorContributor.java

```
package com.fengmanfei.myrpc.editors;

import org.eclipse.jface.action.Action;
import org.eclipse.jface.action.IMenuManager;
import org.eclipse.jface.action.IToolBarManager;
import org.eclipse.jface.action.MenuManager;
import org.eclipse.ui.ISharedImages;
import org.eclipse.ui.PlatformUI;
import org.eclipse.ui.part.EditorActionBarContributor;

public class JsEditorContributor extends EditorActionBarContributor {

    private Action action1 ;
    private Action action2 ;
    public JsEditorContributor() {
        super();
        makeActions();
    }
    public void makeActions() {
        action1 = new Action() {
            public void run() {

            }
        };
        action1.setText("Action 1");
        action1.setToolTipText("Action 1 tooltip");
        action1.setImageDescriptor(PlatformUI.getWorkbench().getSharedImages().
            getImageDescriptor(ISharedImages.IMG_DEF_VIEW));

        action2 = new Action() {
            public void run() {

            }
        };
        action2.setText("Action 2");
        action2.setToolTipText("Action 2 tooltip");
        action2.setImageDescriptor(PlatformUI.getWorkbench().getSharedImages().
            getImageDescriptor(ISharedImages.IMG_OBJS_WARN_TSK));
    }
    //覆盖父类中的方法，创建菜单
    public void contributeToMenu(IMenuManager menuManager) {
        MenuManager editMenu = new MenuManager("编辑器菜单");
        editMenu.add( action1 );
    }
}
```



```

editMenu.add( action2 );
menuManager.add( editMenu );
}
//覆盖父类的方法，创建工具栏
public void contributeToToolBar(IToolBarManager toolBarManager) {
    toolBarManager.add( action1 );
    toolBarManager.add( action2 );
}
}

```

其中，创建菜单栏和工具栏的方法分别是覆盖父类中的 `contributeToMenu` 和 `contributeToToolBar` 方法，而创建 Action 的对象的方法读者应该很熟悉了。

23.3.5 多页编辑器

多页编辑器可以同时打开多个页面，每个页面可以是一个编辑器 `IEditorPart`，也可以是普通控件 `Control`。如图 23.20 和图 23.21 所示为一个多页的编辑器，在每页编辑器的下方还有一个选项卡，可以进行切换。

图 23.20 所示的选项卡页面为一个编辑器 `IEditorPart` 对象，而图 23.21 所示的选项卡页面为一个普通的控件 `Label` 对象。



图 23.20 页面为 `IEditorPart` 对象的多页编辑器

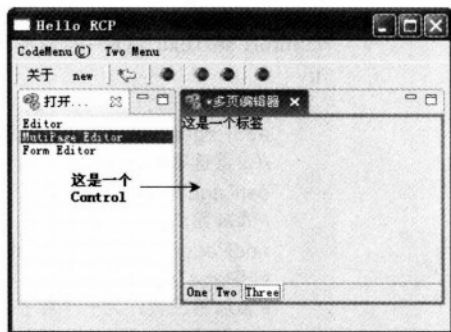


图 23.21 页面 `Control` 对象的多页编辑器

其实实现这样的效果并不难，与上文中创建编辑器类似，只是与创建编辑器所对应的类时的方法稍有不同。下面来具体看一下如何实现。

首先在 `plugin.xml` 配置一个新的编辑器。配置文件如下：

```

<editor
    class="com.fengmanfei.myrpc.editors.MutiEditorSample"
    default="false"
    icon="icons/samples.gif"
    id="com.fengmanfei.myrpc.editors.MutiEditorSample"
    name="多页编辑器"/>

```

其中该编辑器对应的类为 `com.fengmanfei.myrpc.editors.MutiEditorSample`，这里 `MutiEditorSample` 要继承自 `MultiPageEditorPart` 类才可以实现多页显示其效果。

该类的代码如下：

MutiEditorSample.java

```
package com.fengmanfei.myrpc.editors;

import org.eclipse.core.runtime.IProgressMonitor;
import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Label;
import org.eclipse.ui.PartInitException;
import org.eclipse.ui.part.MultiPageEditorPart;

public class MutiEditorSample extends MultiPageEditorPart {
    //该编辑器的标识
    public static final String ID = "com.fengmanfei.myrpc.editors.MutiEditorSample";
    private JsEditor page1 ;//编辑器对象
    private JsEditor page2 ;//编辑器对象
    private Label control1 ;//标签对象
    //父类中抽象方法
    protected void createPages() {
        //创建页面和标签对象
        page1 = new JsEditor();
        page2 = new JsEditor();
        control1 = new Label ( getContainer(), SWT.NONE);
        control1.setText("这是一个标签");
        try {
            //添加第一页
            addPage( page1 , new JsEditorInput("One"));
            //设置选项卡的名称
            setPageText(0,"One");
            //添加第二页
            addPage( page2 , new JsEditorInput("Two"));
            setPageText(1,"Two");
            //添加第三页，为一个标签
            addPage(control1);
            setPageText(2,"Three");
        } catch (PartInitException e) {
            e.printStackTrace();
        }
    }
    public void doSave(IProgressMonitor monitor) {
    }
    public void doSaveAs() {
    }
    public boolean isSaveAsAllowed() {
        return false;
    }
}
```

创建多页编辑器时应该注意以下几个问题：

- ❑ 该类必须继承自 `MultiPageEditorPart` 类。
- ❑ 创建页面的代码是在 `createPages` 方法中。
- ❑ 每个页面可以是 `IEditorPart` 对象，也可以是 `Control` 对象。
- ❑ 创建完页面后要调用 `addPage` 方法，将该页添加。如果不使用该方法，将不会显示页面。
- ❑ `addPage` 方法如下所示：
 - ◆ `addPage(Control control)`: 添加一个控件，其中 `control` 为选项卡中的控件对象。
 - ◆ `addPage(IEditorPart editor, IEditorInput input)`: 添加一个编辑页面。
 - ◆ `addPage(int index, Control control)`: 在指定索引的选项卡中，添加一个控件。其中 `index` 为索引值。
 - ◆ `addPage(int index, IEditorPart editor, IEditorInput input)`: 在指定索引的选项卡中，添加一个编辑页面。其中 `index` 为索引值。
- ❑ 可以使用父类的 `setPageText(int pageIndex, String text)` 方法设定显示选项卡的名称。

23.4 透 视 图

透视图是 Eclipse 平台特有的一种布局窗口的方式。透视图定义“工作台”窗口中视图的初始布局 and 所对应的操作集。例如，Eclipse 平台中每个透视图都提供了一组功能，目的在于完成特定类型的任务或使用特定类型的资源。例如，Java 透视图将编辑 Java 源文件时常用的视图组合在一起，而“调试”透视图包含将在调试 Java 程序时使用的视图。当在“工作台”中工作时，可能会频繁地在各个透视图之间进行切换。

但在 RCP 开发应用程序时，通常不会使用到多个透视图，往往是使用一个默认的视图就足够了。透视图的应用还主要应用在 Eclipse 的平台中。

23.4.1 透视图的布局

透视图是一组操作集合和初始化视图布局的定义，所以要设置配置透视图，首先要理解透视图都可以设置哪些布局。一个透视图中的布局主要元素如图 23.22 所示。

透视图的布局设置是在透视图类的以下所示的方法中创建的：

```
public void createInitialLayout(IPageLayout layout)
```

在第 22 章中曾经提到过，例如本例中默认的透视图类为 `Perspective` 类，读者可以在该类中设置透视图的布局。而透视图的布局主要是通过 `IPageLayout` 对象来设置的。

(1) 首先，一个透视图只有一个编辑区域 (`EditorArea`)，如图 23.22 所示的中间部分即为编辑区域。可以通过 `IPageLayout` 对象的 `getEditorArea()` 方法获得，例如以下代码：

```
String editorArea = layout.getEditorArea();
```

如果想不显示编辑区域可以使用以下方法来隐藏编辑区域：

`setEditorAreaVisible(boolean showEditorArea)`

(2) 设置操作集 (ActionSet)，可以通过以下方法来设置：

`addActionSet(String actionSetId)`

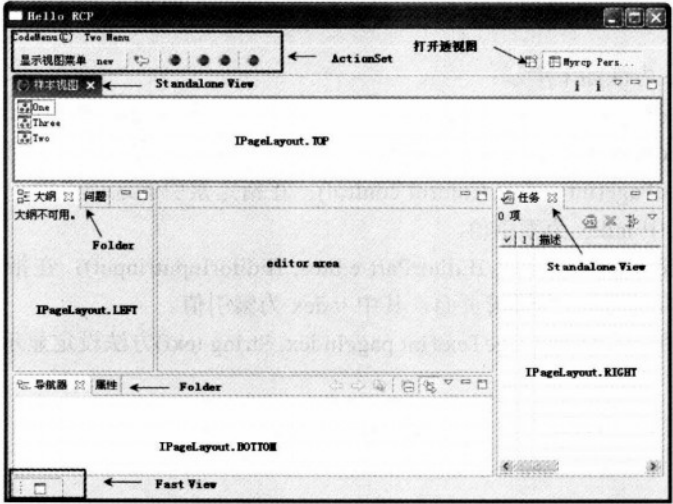


图 23.22 透视图的布局示意图

其中，`actionSetId` 为扩展 `org.eclipse.ui.actionSet` 操作集中所使用的 id 标识。

(3) 最后设置视图布局，对于这个窗口区域来说，整体上分为上(TOP)、下(BOTTOM)、左(LEFT)、右(RIGHT) 4 个部分，每个部分又可以添加不同视图的类型。以下列举了 `IPageLayout` 对象中设置视图布局的几种方法：

- ❑ `addStandaloneView(String viewId, boolean showTitle, int relationship, float ratio, String refId)`：添加独立的视图。

`Standalone View` 表示这个视图不能与其他视图重叠，例如图 23.23 所示的上方的“样本视图”部分，只能显示一个视图。而且这种视图拖动的过程中也不会与其他的视图重叠，只独立显示的视图。以下是这几个参数所表示的意义：



图 23.23 样本视图

- ◆ `viewId`：表示视图所对应的标识 id。

- ◆ **showTitle**: 表示是否显示标签和关闭按钮。例如, 若设置为 `false` 时, 视图的界面效果如图 23.24 所示。

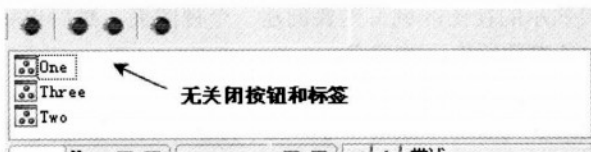


图 23.24 无标签和关闭按钮时的视图效果

- ◆ **relationship**: 表示设置的位置, 值可以是 `IPageLayout.TOP`、`IPageLayout.BOTTOM`、`IPageLayout.LEFT` 和 `IPageLayout.RIGHT` 分别表示上下左右的位置。
- ◆ **Ratio**: 表示所占区域的百分比, 值可以是 0.05~0.95 之间任意的值。例如, 0.25 表示所占的比例为 25%。
- ◆ **refId**: 相对的标识, 可以是相对于视图的 `id` 或者视图夹的 `id` 或者是编辑区域的 `id`。

例如, 本例中使用的都是相对于编辑区域的位置, 本例中上方和右方创建的就是独立的视图, 代码如下:

```
layout.addStandaloneView(SampleView.ID,false,IPageLayout.TOP,0.45f,editorArea);
layout.addStandaloneView(IPageLayout.ID_TASK_LIST,false,IPageLayout.RIGHT,0.45f,editorArea);
```

若将右侧的放置视图的相对 `id` 换成 `SampleView.ID`, 改成如下所示的代码, 则第二个视图将显示在 `SampleView.ID` 视图的右侧, 如图 23.25 所示。

```
layout.addStandaloneView(SampleView.ID,true,IPageLayout.TOP,0.45f,editorArea);
layout.addStandaloneView(IPageLayout.ID_TASK_LIST,true,IPageLayout.RIGHT,0.45f,SampleView.ID);
```

图 23.25 相对位置改为 `SampleView.ID` 后的效果

- **addView(String viewId, int relationship, float ratio,String refId)**: 普通的视图。

设置一个普通的视图, 与 `Standalone View` 视图不一样, 该视图可以拖动到其视图夹中, 而不会独立显示。其中的参数与 `addStandaloneView` 方法中的参数意义相同。

- **addPlaceholder(String viewId, int relationship, float ratio,String refId)**: 预置一个视图空间。

该方法添加的视图不会显示在界面上, 但是当通过操作打开该设定的视图时, 将会在该设定的位置打开, 而不是随意打开该视图。简单地说就是为指定的视图预留一个空间, 一旦视图打开就在该空间中显示。其中各参数与 `addView` 方法中的参数意义相同。

- **IFolderLayout createFolder(String folderId, int relationship, float ratio, String refId)**: 添加可重叠的视图。

工作区中使用最多的还是可重叠显示的视图，如图 23.22 所示，左侧和下方的视图即为可重叠的视图。一个这样的视图区域称之为视图夹。一个视图夹中只能激活一个视图。所以要显示一个可多层显示的视图区域首先要创建一个视图夹，然后再将视图添加到视图夹中。以下是该方法中各参数所表示的意义：

- ◆ folderId: 在一个透视图区域要唯一标识。
- ◆ relationship、ratio 和 refId: 与上两个方法中的参数意义相同。

例如本例中，创建左侧和下方的视图夹的代码如下：

```
IFolderLayout bottom = layout.createFolder("bottom", IPageLayout.BOTTOM, 0.25f, editorArea);
bottom.addView(IPageLayout.ID_RES_NAV);
bottom.addView(IPageLayout.ID_PROP_SHEET);
bottom.addPlaceholder(IPageLayout.ID_BOOKMARKS);

IFolderLayout left = layout.createFolder("left", IPageLayout.LEFT, 0.25f, editorArea);
left.addView(IPageLayout.ID_OUTLINE);
left.addView(IPageLayout.ID_PROBLEM_VIEW);
```

□ addFastView(String viewId)和 addFastView(String viewId, float ratio): 设置快速视图。快速视图是显示在状态栏左侧的区域中，一般可以将经常使用的视图放置到快速视图中。如图 23.22 中左下角的区域。本例中设置快速视图的代码如下：

```
layout.addFastView(AnotherView.ID);
```

但要注意，在显示快速视图时，在创建初始化窗口时要设置快速视图栏可见才可以。例如，本例中需要在 ApplicationWorkbenchWindowAdvisor 类的 preWindowOpen 方法中添加以下代码：

```
public void preWindowOpen() {
    IWorkbenchWindowConfigurer configurer = getWindowConfigurer();
    //.....代码省略
    configurer.setShowFastViewBars(true); //设置显示快速视图
}
```

23.4.2 透视图扩展点

一个 RCP 程序中必须至少有一个默认的视图。当前也可以添加多个视图，这就需要扩展 org.eclipse.ui.perspectives 扩展点。

下面就看一下如何扩展透视图的方法。首先找到扩展透视图的扩展点 org.eclipse.ui.perspectives，配置扩展信息。配置后的 plugin.xml 文件如下：

```
<extension
    point="org.eclipse.ui.perspectives">
    <perspective
        class="com.fengmanfei.myrpc.MyPerspective"
        icon="icons/samples.gif"
        id="com.fengmanfei.myrpc.MyPerspective"
```

```
name="自定义透视图"/>
</extension>
```

其中，透视图扩展点的各元素的说明如下：

- ❑ 透视图的扩展点类型为 `org.eclipse.ui.perspectives`。
- ❑ `class` 为该透视图所对应的透视图类，该类必须是实现 `org.eclipse.ui.IperspectiveFactory` 的类。
- ❑ `name` 为透视图的名称，在打开透视图时显示，如图 23.26 所示。

23.4.3 透视图类

下面来看一下透视图所对应的类 `com.fengmanfei.myrpc.MyPerspective` 中的代码。代码很简单，如下：

MyPerspective.java

```
package com.fengmanfei.myrpc;

import org.eclipse.ui.IPageLayout;
import org.eclipse.ui.IPerspectiveFactory;
import com.fengmanfei.myrpc.views.AnotherView;
import com.fengmanfei.myrpc.views.SampleView;

public class MyPerspective implements IPerspectiveFactory {

    public void createInitialLayout(IPageLayout layout) {
        String editorArea = layout.getEditorArea();
        layout.addStandaloneView(SampleView.ID, true, IPageLayout.LEFT, 0.45f, editorArea);
        layout.addStandaloneView(AnotherView.ID, true, IPageLayout.BOTTOM, 0.45f, editorArea);
    }
}
```

这样选择该透视图后，界面的效果如图 23.27 所示。

需要注意的，如果想要使工具栏中显示选择视图的按钮，如图 23.28 所示，则需要与显示快速视图的方法一样设置显示选择透视图按钮。

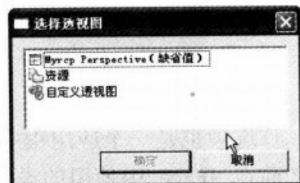


图 23.26 打开透视图对话框

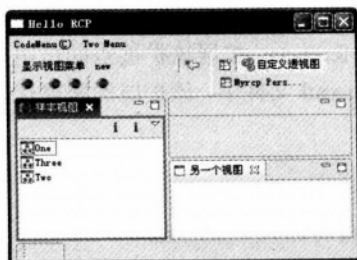


图 23.27 自定义的透视图布局

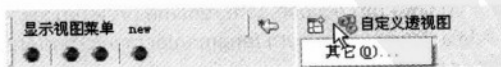


图 23.28 选择透视图

例如, 修改 `ApplicationWorkbenchWindowAdvisor` 类中的代码如下:

```
public void preWindowOpen() {
    IWorkbenchWindowConfigurer configurer = getWindowConfigurer();
    //.....代码省略
    configurer.setShowPerspectiveBar(true); //设置显示透视图按钮
}
```

23.5 首 选 项

首选项也是一个应用程序中不可缺少的一部分。开发 RCP 程序时, 开发首选项的方法与之前在 JFace 中使用的设置首选项的方法没有什么大的变化, 只是配置的方法不同。如图 23.29 所示即为 RCP 程序中开发的首选项的界面效果。

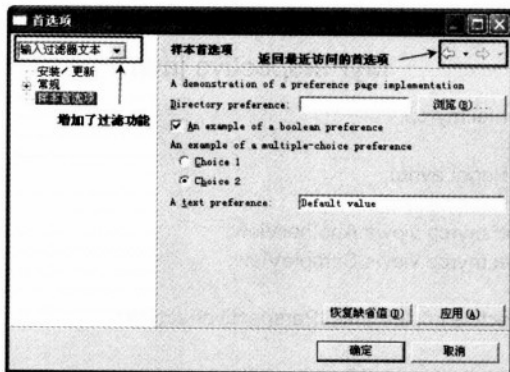


图 23.29 RCP 开发的首选项界面

从图 23.29 中可以看出, 与 JFace 中的首选项不同之处是在导航页面增加了筛选的功能, 另外在首选项的上方, 增加了返回和前进的按钮。

与之前使用的扩展点不同, 扩展首选项时要使用两个扩展点, 分别是 `org.eclipse.core.runtime.preferences` 和 `org.eclipse.ui.preferencePages`。

23.5.1 首选项扩展点

扩展首选项首先要扩展 `org.eclipse.core.runtime.preferences`, 扩展该扩展点的作用是初始化保存首选项的文件和设置首选项的默认值。在 `plugin.xml` 文件中, 该扩展的配置代码很简单, 如下:

```
<extension
    point="org.eclipse.core.runtime.preferences">
    <initializer class="com.fengmanfei.myrpc.preferences.PreferenceInitializer"/>
</extension>
```

其中<initializer>中 class 表示初始化首选项时所装载的类名。该类通常是继承 org.eclipse.core.runtime.preferences.AbstractPreferenceInitializer 抽象类。

例如, 本例中对应的类为 com.fengmanfei.myrpc.preferences.PreferenceInitializer。以下为该类的具体代码:

PreferenceInitializer.java

```
package com.fengmanfei.myrpc.preferences;

import org.eclipse.core.runtime.preferences.AbstractPreferenceInitializer;
import org.eclipse.jface.preference.IPreferenceStore;
import myRCP.MyRCPPlugin;

public class PreferenceInitializer extends AbstractPreferenceInitializer {

    public void initializeDefaultPreferences() {
        //获得该插件保存的首选项保存对象
        IPreferenceStore store = MyRCPPlugin.getDefault().getPreferenceStore();
        //设置默认的一些首选项值
        store.setDefault(PreferenceConstants.P_BOOLEAN, true);
        store.setDefault(PreferenceConstants.P_CHOICE, "choice2");
        store.setDefault(PreferenceConstants.P_STRING, "Default value");
    }
}
```

需要注意的地方就是获得该插件所在的首选项保存对象的方法是:

```
MyRCPPlugin.getDefault().getPreferenceStore()
```

其中, 将首选项使用的一些 key 保存在了 PreferenceConstants 的常量中。该类的代码如下:

PreferenceConstants.java

```
package com.fengmanfei.myrpc.preferences;

public class PreferenceConstants {

    public static final String P_PATH = "pathPreference";
    public static final String P_BOOLEAN = "booleanPreference";
    public static final String P_CHOICE = "choicePreference";
    public static final String P_STRING = "stringPreference";

}
```

23.5.2 首选项页面扩展点

有了初始化首选项后, 还需要向首选项页面添加选项页面才可以显示首选项, 每个首选项页面都要扩展 org.eclipse.ui.preferencePages 扩展点。本示例程序中的 plugin.xml 文件中, 该扩展的配置代码如下:

```
<extension
```

```

        point="org.eclipse.ui.preferencePages">
    <page
        class="com.fengmanfei.myrpc.preferences.SamplePreferencePage"
        id="com.fengmanfei.myrpc.preferences.SamplePreferencePage"
        name="样本首选项">
        <keywordReference id="样本"/>
    </page>
</extension>

```

其中，首选项页面扩展点的各元素的说明如下所示：

- ❑ 首选项页面扩展点类型为 `org.eclipse.ui.preferencePages`。
- ❑ `class` 为对应的该首选项页面所对应的 Java 类的名称，该类必须是实现了 `org.eclipse.ui.IWorkbenchPreferencePage` 接口的类，通常使用时是继承 `FieldEditorPreferencePage` 类（与 JFace 中使用方法相同）并实现 `IWorkbenchPreferencePage` 接口。
- ❑ `<keywordReference>` 表示该页面搜索时的关键字。
例如，在搜索文本框中输入本例页面中设置的关键字后，会显示该页面，效果如图 23.30 所示。
- ❑ 本例中首选项页面另外还有一个 `<category>` 元素，可设置该首选项页面在导航栏中树型结构的位置。

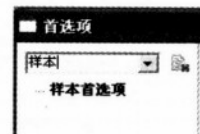


图 23.30 输入关键字后的效果

本例中该首选项页面所对应的类为 `SamplePreferencePage`。该类的代码如下：

SamplePreferencePage.java

```

package com.fengmanfei.myrpc.preferences;

import org.eclipse.jface.preference.*;
import org.eclipse.ui.IWorkbenchPreferencePage;
import org.eclipse.ui.IWorkbench;
import org.eclipse.ui.plugin.MyRCPPlugin;

public class SamplePreferencePage extends FieldEditorPreferencePage implements
    IWorkbenchPreferencePage {

    public SamplePreferencePage() {
        super(GRID);
        //设置首选项保存对象
        setPreferenceStore(MyRCPPlugin.getDefault().getPreferenceStore());
        setDescription("A demonstration of a preference page implementation");
    }

    //创建字段编辑器
    public void createFieldEditors() {
        addField(new DirectoryFieldEditor(P_PATH,
            "&Directory preference:", getFieldEditorParent()));
        addField(new BooleanFieldEditor(P_BOOLEAN,
            "&An example of a boolean preference", getFieldEditorParent()));
        addField(new RadioGroupFieldEditor(P_CHOICE,

```



```
"An example of a multiple-choice preference", 1,
new String[] { "&Choice 1", "choice1" },
    { "C&hoice 2", "choice2" } }, getFieldEditorParent());
addField(new StringFieldEditor(PreferenceConstants.P_STRING,
    "A &text preference:", getFieldEditorParent()));
}
// IWorkbenchPreferencePage 中的方法, 暂时空实现
public void init(IWorkbench workbench) {
}
}
```

以上代码读者不陌生, 与在 JFace 中使用的字段编辑器方式创建的首选项的方法是一样的, 只不过这里实现了 IWorkbenchPreferencePage 接口。

最后, 要调用出首选项页面需要调用 Eclipse 内置的 Action, 代码如下所示:

```
IWorkbenchAction refOpen = ActionFactory.PREFERENCES.create(window)
```

23.6 帮助文档

帮助文档也是一个应用系统所必须有的一项内容之一, Eclipse 已经提供了一套完善的添加帮助文档的框架。

帮助的方式有多种, 一种是打开帮助文档浏览界面, 在 Eclipse 中是选择“帮助”|“帮助内容”命令来查看的。另一种方式是上下文的帮助, 当在某个焦点时按 F1 键获得所需要的上下文帮助。如图 23.31 所示, 即为在 Eclipse 工作区中使用的上下文帮助的界面示意图。



图 23.31 Eclipse 工作区中所使用的上下文帮助

后效果如图 23.35 所示。



图 23.34 导入联机帮助的依赖项



图 23.35 联机帮助窗口打开后的效果图

23.6.2 扩展配置

下面来具体分析一下配置文件中是如何进行配置的。以下为扩展该联机帮助后 plugin.xml 文件中所生成的配置信息。

```
<extension
    point="org.eclipse.help.toc">
    <toc
        file="toc.xml"
        primary="true"/>
    <toc file="tocconcepts.xml"/>
    <toc file="tocgettingstarted.xml"/>
    <toc file="tocreference.xml"/>
    <toc file="tocsamples.xml"/>
    <toc file="toctasks.xml"/>
</extension>
```

下面来说明各个元素的含义：

- ❑ 联机帮助扩展点的类型为 org.eclipse.help.toc。
- ❑ 每一个联机帮助的主题都是以一个 < toc>元素开始的。
- ❑ <toc>元素中的 primary 表示，该文档是否是根目录中的帮助。
- ❑ <toc>元素中的 file 元素表示所对应的一个 xml 文件，记录了浏览帮助的导航规则。要符合一个格式，是最重要的文件。其中这些文件所在路径即为项目所在工作区的路径中，例如本例中这些文件所在项目中文件的位置如图 23.36 所示。

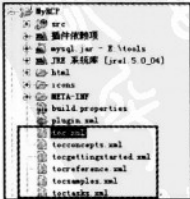


图 23.36 配置文件中文档所在的路径

23.6.3 联机帮助的目录结构

有了配置的扩展信息，最重要的是要理清帮助文档页面的关系。首先来看一下本例中根目录的配置文件 toc.xml 中配置代码如下：

toc.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?NLS TYPE="org.eclipse.help.toc"?>

<toc label="Sample Table of Contents" topic="html/toc.html">
  <topic label="Getting Started">
    <anchor id="gettingstarted"/>
  </topic>
  <topic label="Concepts">
    <anchor id="concepts"/>
  </topic>
  <topic label="Tasks">
    <anchor id="tasks"/>
  </topic>
  <topic label="Reference">
    <anchor id="reference"/>
  </topic>
  <topic label="Samples">
    <anchor id="samples"/>
  </topic>
</toc>
```

这个是帮助目录中根目录的配置文件，读者可以结合图 23.34 来理解配置代码的意义。配置帮助目录时要注意以下问题：

- ❑ 要声明文档的类型为<?NLS TYPE="org.eclipse.help.toc"?>。
- ❑ toc 元素中，label 表示所显示的标签名称，topic 表示该导航页所显示的网页文件，注意，文件的路径结构。本例中所引用的是 html/toc.html，那么该文件在项目中的结构目录如图 23.37 所示。



图 23.37 帮助网页的目录结构

- ❑ topic 元素中，anchor 的 id 表示该目录所连接的锚元素，可供其他的链接引用。本

例中注意 anchor 的 id 为 gettingstarted 的锚点。
然后再来看一下 tocgettingstarted.xml 这个文件中的配置代码，如下。

tocgettingstarted.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?NLS TYPE="org.eclipse.help.toc"?>

<toc label="Getting Started" link_to="toc.xml#gettingstarted">
  <topic label="Main Topic" href="html/gettingstarted/maintopic.html">
    <topic label="Sub Topic" href="html/gettingstarted/subtopic.html" />
  </topic>
  <topic label="Main Topic 2">
    <topic label="Sub Topic 2" href="html/gettingstarted/subtopic2.html" />
  </topic>
</toc>
```

从该配置的代码中可以看到，该主题中 link_to 元素的值为 "toc.xml#gettingstarted"，正是 toc.xml 文件中锚点为 gettingstarted，所以这个主题文件中的配置附加到 gettingstarted 锚点中。也就是说当单击 gettingstarted 锚点所对应的链接后，会显示该主题中配置的文档目录。

该目录下又有多个 <topic>，每个 <topic> 可以对应一个 href，表示单击该 topic 时，所指向的网页页面。

23.6.4 添加动态帮助

除了联机帮助外也可以为界面中的元素添加动态的帮助，动态帮助是通过按 F1 键来获得的，要添加动态帮助，需要扩展 org.eclipse.help.contexts 扩展点。以下为添加动态帮助的步骤：

- (1) 在 plugin.xml 文件中配置以下代码：

```
<extension
  point="org.eclipse.help.contexts">
  <contexts file="SampleViewHelp.xml"/>
</extension>
```

其中，file 的文件名为设置动态帮助的 xml 文件，该文件要符合一个格式。

- (2) 新建一个所对应的 SampleViewHelp.xml 文件，然后输入以下代码：

```
<?xml version="1.0" encoding="UTF-8"?>
<contexts>
  <context id="SampleViewContextId">
    <description> 这是一个帮助示例.</description>
    <topic href="html/samples/maintopic.html" label="主题 1"/>
    <topic href="html/samples/subtopic.html" label="主题 2"/>
  </context>
</contexts>
```


其中, id 很重要, 为该上下文帮助的唯一标识。

(3) 最后, 为该帮助指定所关联的控件, 例如在上文视图中, 有一个 `SampleView` 的类, 可以在该类的方法 `createPartControl` 中添加以下代码:

```
PlatformUI.getWorkbench().getHelpSystem().setHelp(viewer.getTable(), "SampleViewContextId");
```

其中 `PlatformUI.getWorkbench().getHelpSystem()` 方法可以获得帮助对象 `IWorkbenchHelpSystem`。然后将上文中定义的 context 的 id 注册到帮助系统中。

`IWorkbenchHelpSystem` 对象除了可以为某个控件注册上下文帮助外, 也可以为菜单、操作注册上下文帮助。该类中的主要方法如下所示:

- ☐ `setHelp(IAction action, String contextId)`: 为操作注册上下文帮助。
- ☐ `setHelp(Control control, String contextId)`: 为控件注册上下文帮助。
- ☐ `setHelp(Menu menu, String contextId)`: 为菜单注册上下文帮助。
- ☐ `setHelp(MenuItem item, String contextId)`: 为菜单项注册上下文帮助。

23.7 RCP 产品的发布

RCP 程序开发完成后, 需要打包发布给用户使用。对于开发人员来说可以使用 Eclipse 平台来运行 RCP 的程序, 可最终使用的用户不可能了解什么是 Eclipse, 他们只关注的是这个产品能不能用的问题。所以这就涉及将 RCP 产品发布的问题。

幸运的是, RCP 产品发布的功能非常方便, 使用 Eclipse 的产品配置就可以快速地打包 RCP 产品。下面就来介绍如何将 RCP 发布成产品供用户使用。

23.7.1 Eclipse 产品配置

产品配置中包括程序的名称、图标、关于产品的内容等信息, 是打包 RCP 产品中重要的配置文件。以下为创建产品配置文件的步骤:

(1) 选择“文件”|“新建”|“产品配置”命令, 将弹出“新建产品配置”对话框, 如图 23.38 所示。然后输入产品配置的文件名, 并选中“使用现有产品”单选按钮, 并且选中本例中所创建的 RCP 项目。

(2) 单击“完成”按钮, 进入到产品配置的页面, 在该页面中主要有“概述”、“配置”和“标记”3 个页面。如图 23.39 所示为“概述”页面。

(3) 按照图 23.39 所示的页面配置好产品的信息。然后选择“配置”选项卡, 进入到配置页面, 如图 23.40 所示。这里可以查看该产品所包含的一些插件和段信息, 并且右侧也可以指定启动程序的变量。本例中不多作设置, 使用默认设置就可以了。

(4) 在“标记”页面中输入一些自定义的信息, 如图 23.41 所示。需要注意的是, 需要将这些使用的图片放置在项目的文件夹中。

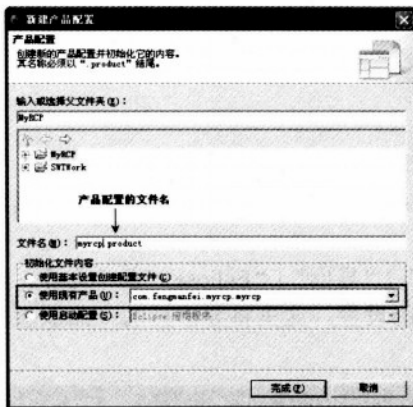


图 23.38 “新建产品配置”页面

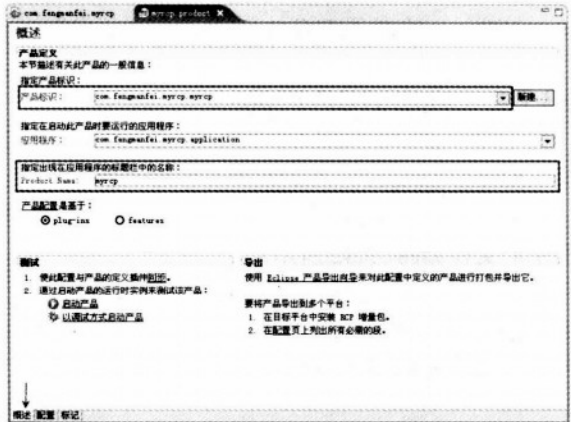


图 23.39 “概述”页面



图 23.40 “配置”页面



图 23.41 “标记”页面的详细配置

这样，一个基于 Eclipse 产品的配置就设置好了，接下来就可以导出 RCP 应用程序了。

23.7.2 导出 RCP 产品

配置好 Eclipse 产品的配置文件后，就可以导出产品了。具体的步骤如下：

(1) 在导出配置文件之前，首先要在构建文件中确保导出时所包含的所有文件，具体选中的文件如图 23.42 所示。

(2) 然后就可以导出 RCP 程序了，选择“文件”|“导出”|“Eclipse 产品”命令，弹出如图 23.43 所示的“导出”对话框，并进行详细的设置。这里将导出后的产品放置在 E:\RcpProduct\RCP 产品发布\文件夹下。



图 23.42 构建文件（build.properties）的设置

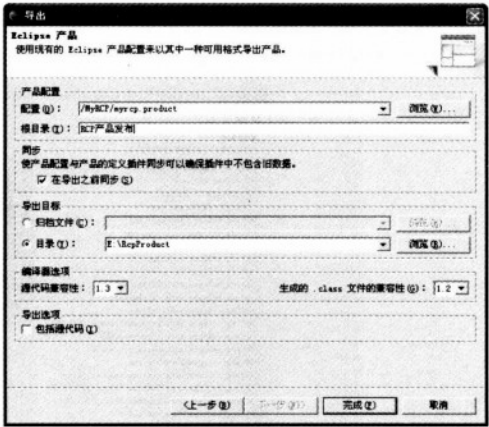


图 23.43 “导出”对话框

(3) 最后单击“完成”按钮，这样一个产品就发布完毕了。

23.7.3 运行 RCP 产品

最后看一下最后导出后的产品究竟是什么样子的。

(1) 本例中导出的路径为 E:\RcpProduct\RCP 产品发布，这样打开该文件夹，可以看到该文件夹中的文件如图 23.44 所示。

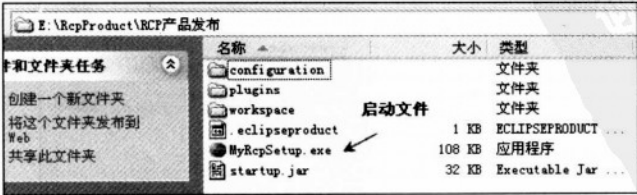


图 23.44 发布后的 RCP 产品

(2) 双击这个 exe 文件，就可以运行 RCP 程序了。当然前提条件是该系统已经安装了

Java 虚拟机。再来查看一下文件夹的大小，只有 20.6 MB，只使用 Eclipse 的一些类包就可以了。

(3) 如图 23.45 和图 23.46 所示即为最终该 RCP 产品打包后的界面效果。

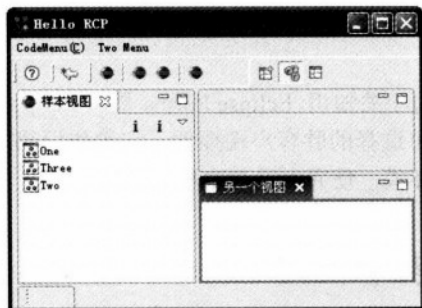


图 23.45 主页面

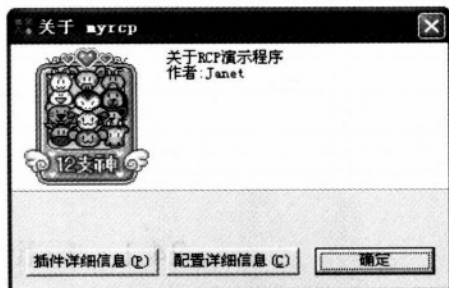


图 23.46 关于页面

当然，交付用户使用时还需要使用一些打包软件，如 InstallShield 软件，打包成安装文件，这些内容就不详细介绍了。

23.8 本章小结

本章主要是对 RCP 开发中常用的一些扩展作了详细的介绍。了解了这些扩展点，能满足大部分程序开发中的需要，但 Eclipse 插件开发中的扩展点还有很多，如语言的本地化、定制界面的样式等，不能一一地详细讲述。Eclipse 的帮助文档中有这些扩展点的详细说明。RCP 的出现使开发桌面程序 SWT 和 JFace 变得更加轻松，所以是很好的开发 SWT 程序的框架。

最后，在实际的开发项目的过程中可能会遇到各种各样的问题，还需要读者自己去解决，所以要养成阅读源代码的习惯。可以多尝试阅读一些 Eclipse 的源代码，因为 Eclipse 本身的源代码就是很好的学习资料。



第 24 章 Eclipse 表单

本章将详细讲述 Eclipse 表单(Eclipse Forms)的相关知识。Eclipse Forms 是从 Eclipse 3.0 版后添加的新特性,是一个基于 SWT 与 JFace 的可选择的胖客户端插件,它的作用就是为 SWT/JFace 的各种控件渲染成类似于 Web 表单的样式,使界面看起来更加精致。

24.1 Eclipse 表单概述

使用 Eclipse 表单的组件包,可以使 SWT/JFace 的各种控件呈现出类似于网页中的效果,对用户来说,是一种全新的体验。

24.1.1 什么是 Eclipse 表单

Eclipse 表单本身是以插件的形式发布的,由一组定制的小部件和支持类组成。在 Eclipse 3.0 版本以前只是在 Eclipse 的插件开发环境(PDE)以及 Eclipse 插件升级与更新(Update)组件内部使用。PDE 是 Eclipse 为插件开发人员提供的一套支持与辅助工具集,其中最常见就是在讲述 RCP 开发时所用到的插件清单 plugin.xml 文件,这就是一个典型的 Eclipse 表单的应用。如图 24.1 所示为 plugin.xml 清单所对应的表单页面。

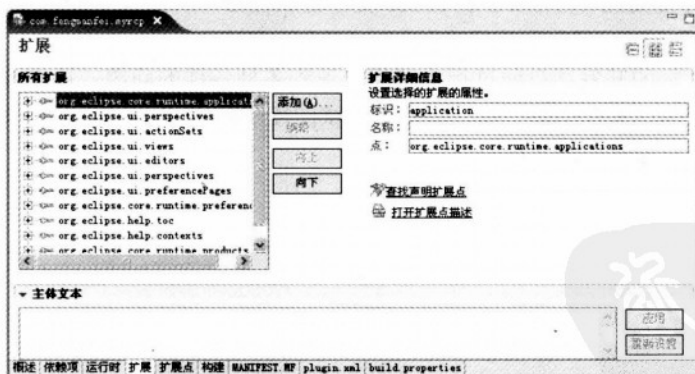


图 24.1 Eclipse 表单应用

表单是一个基于 SWT 与 JFace 的可选择的胖客户端插件,它的作用就是为用户提供一套精美的类似于 Web 网页设计中的一些用户接口。Eclipse 的开发者在实现表单时,非常谨慎地使这些用户接口,从操作类型、颜色、字体等属性扩展 SWT 以达到预期的效果。正因为是基于 SWT 开发而成的,所以也就具备了 SWT 的跨平台特性。

24.1.2 Eclipse 表单的特性

表单可以应用于 Eclipse 工作台的任意一个 UI 类别当中，如视图、编辑器、向导、对话框等。以下列出了 Eclipse 表单的部分特性，这些特性使 SWT/JFace 的控件呈现出类似于 Web 网页的精美 UI 界面。

- ❑ 提出了表单的概念，可以应用到编辑器、视图等容器中，为开发 UI 界面提供了另一种实现界面效果的方式。
- ❑ 使用一个专有的表单工具包，以抽象工厂的设计模式创建颜色管理器、超链接和其他类似 SWT 控件的表单控件。
- ❑ 提供了类似使用 HTML 表格布局的新布局管理器 `TableLayout`。
- ❑ 提供了很多专门为表单设计的定制控件，包括超链接、图像链接、可滚动的面板等。
- ❑ 可以绑定一个多页编辑器，使其每页是一个包含表单的页，如 `plugin.xml` 清单编辑器。

24.1.3 Eclipse 表单使用的类包

要开发 Eclipse 表单，首先要确保在构建路径中导入了以下两个类包：

- ❑ `org.eclipse.ui.forms_3.1.0.jar`：表单的 jar 包。
- ❑ `org.eclipse.ui.forms.nl_3.1.0.jar`：表单的语言包（可选）。

24.2 表单开发基础

在开发表单之前，首先了解一下开发表单的一般过程，然后再讲述具体的每个不同的表单控件。

24.2.1 视图中使用表单

表单不仅可以应用在编辑器中（例如，之前所说的在 `plugin.xml` 中使用的编辑器），也可以应用在视图中，当然也可以应用在其他面板容器中。这里就以视图为例，来看一下如何在视图中使用表单。

如图 24.2 所示为在视图中创建的一个简单的表单示意图。

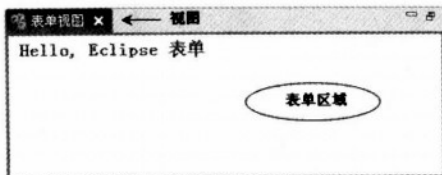


图 24.2 视图中的表单

(1) 按照创建视图的方式扩展一个视图扩展点。设置视图扩展后的 plugin.xml 文件中的代码如下:

```
<extension
    point="org.eclipse.ui.views">
    <view
        class="com.fengmanfei.myrpc.views.MyFormView"
        icon="icons/samples.gif"
        id="com.fengmanfei.myrpc.views.MyFormView"
        name="表单视图"/>
    </extension>
```

(2) 从以上的配置文件中可以看出, 该视图所对应的类为 com.fengmanfei.myrpc.views.MyFormView。创建表单的代码就是在该类中实现的, 以下为该类具体实现的代码:

MyFormView.java

```
package com.fengmanfei.myrpc.views;

import org.eclipse.swt.widgets.Composite;
import org.eclipse.ui.forms.widgets.FormToolkit;
import org.eclipse.ui.forms.widgets.ScrolledForm;
import org.eclipse.ui.part.ViewPart;

public class MyFormView extends ViewPart {

    public static final String ID = "com.fengmanfei.myrpc.views.MyFormView";
    private FormToolkit toolkit; // 表单的工具对象
    private ScrolledForm form; // 可滚动的表单对象
    public MyFormView() {
        super();
    }
    // 实现父类中的抽象方法, 创建视图中的各种控件
    public void createPartControl(Composite parent) {
        // 创建表单工具对象
        toolkit = new FormToolkit(parent.getDisplay());
        // 通过表单工具对象创建可滚动的表单对象
        form = toolkit.createScrolledForm(parent);
        // 设置表单文本
        form.setText("Hello, Eclipse 表单");
    }
    // 视图获取焦点时
    public void setFocus() {
        form.setFocus(); // 将焦点放置在表单上
    }
    // 覆盖父类中的方法, 释放视图时
    public void dispose() {
        toolkit.dispose(); // 释放表单工具对象
        super.dispose();
    }
}
```

这样，就实现了在视图中创建一个简单的表单。但是，在使用复杂的表单之前还需要注意以下几个问题：

- ❑ **FormToolkit (表单工具)** 对象非常重要，它是创建各种表单控件的中介。简单地说，各种控件都需要通过该对象包装一下，然后渲染出新的样式。本例中创建 **FormToolkit** 对象的代码如下：

```
toolkit = new FormToolkit(parent.getDisplay());
```

- ❑ 通过 **FormToolkit** 对象的 **createScrolledForm** 方法，可以创建一个 **ScrolledForm** 表单对象。读者在这里要熟悉这种通过 **FormToolkit** 对象创建控件的方法，下文中创建表单的各种控件都是通过这种方式创建的。例如，创建表单标签的方法是 **createLabel** 等。
- ❑ 最后，需要注意的是，**FormToolkit** 对象携带了系统的资源，如字体、颜色等，所以在使用完后需要释放，本例中为释放该视图时就释放 **FormToolkit** 对象。

24.2.2 多页编辑器中使用表单

除了在视图中可以使用表单外，编辑器中也可以使用表单。对于一个页面的编辑器与视图中使用表单相同，只需在创建编辑器内容的 **createPartControl** 方法中创建表单就可以了。

这里需要讲述的是如何创建多页的表单编辑器。在第 23 章的图 23.14 所示的编辑器类的继承关系示意图中可以看到，**FormEditor** 继承自 **MultiPageEditorPart**，是显示多页表单的编辑器。如图 24.3 所示，即为一个多页的表单编辑器。

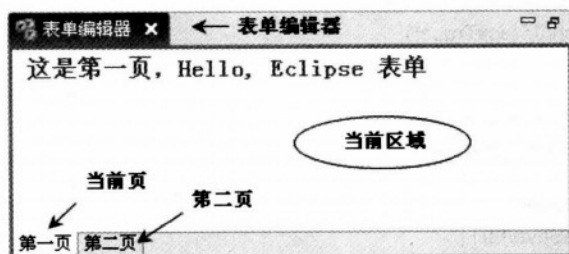


图 24.3 多页的表单编辑器

下面就来看一下如何在多页编辑器中使用表单。

- (1) 按照配置编辑器的方式在 **plugin.xml** 文件中增加一个编辑器扩展。配置后的代码如下：

```
<extension
    point="org.eclipse.ui.editors">
    <editor
        class="com.fengmanfei.myrpc.forms.MyMutiForm"
        default="false"
        icon="icons/samples.gif"
        id="com.fengmanfei.myrpc.forms.MyMutiForm"
```

```
name="表单编辑器"/>
</extension>
```

(2) 从以上的配置代码中可以看出, 该编辑器所对应的编辑器类为 `com.fengmanfei.myrpc.forms.MyMutiForm` 类, 该类继承自 `FormEditor`。具体代码如下:

MyMutiForm.java

```
package com.fengmanfei.myrpc.forms;

import org.eclipse.core.runtime.IProgressMonitor;
import org.eclipse.ui.PartInitException;
import org.eclipse.ui.forms.editor.FormEditor;

public class MyMutiForm extends FormEditor {

    public static final String ID = "com.fengmanfei.myrpc.forms.MyMutiForm";
    public MyMutiForm() {
        super();
    }
    //实现父类中的抽象方法
    //在该方法中创建每个页面
    protected void addPages() {
        try {
            //添加两个页面
            addPage(new FirstPage(this));
            addPage(new SecondPage(this));
        } catch (PartInitException e) {
            e.printStackTrace();
        }
    }
    //保存该编辑器时调用该方法
    public void doSave(IProgressMonitor monitor) {
    }
    //另存为该编辑器时
    public void doSaveAs() {
    }
    //是否允许保存
    public boolean isSaveAsAllowed() {
        return false;
    }
}
```

从以上代码可以看出, 与 23.3.5 节中所创建的多页编辑器没什么不同, 只是两个类继承的父类不同, 这里继承的是 `FormEditor` 类。

(3) 创建每个表单编辑器页。在前面的章节中已经讲述过多页编辑器的实现, 除了编辑器类之外, 还需要一些编辑器的页。在表单的多页编辑器中也是如此, 但所不同的是每个表单页面可以通过继承 `FormPage` 类, 例如, 本例中所使用的 `FirstPage` 和 `SecondPage` 两个类都是继承自 `FormPage` 的。下面就来具体看一下这两个类的代码, `FirstPage` 的实现代码

如下:

FormPage.java

```
package com.fengmanfei.myrpc.forms;

import org.eclipse.ui.forms.IManagedForm;
import org.eclipse.ui.forms.editor.FormEditor;
import org.eclipse.ui.forms.editor.FormPage;
import org.eclipse.ui.forms.widgets.FormToolkit;
import org.eclipse.ui.forms.widgets.ScrolledForm;

public class FirstPage extends FormPage {

    public static final String ID = "com.fengmanfei.myrpc.forms.FirstPage";
    public FirstPage(FormEditor editor) {
        //构造方法, 设置表单页的 ID 和名称
        super(editor, ID, "第一页");
    }
    //覆盖父类中的方法
    //在该方法中创建表单区域的各种控件
    protected void createFormContent(IManagedForm managedForm) {
        //通过 managedForm 对象获得表单工具对象
        FormToolkit toolkit=managedForm.getToolkit();
        //通过 managedForm 对象获得 ScrolledForm 可滚动的表单对象
        ScrolledForm form=managedForm.getForm();
        //设置表单文本
        form.setText("这是第一页, Hello, Eclipse 表单");
    }
}
```

通过以上表单页面的代码, 可以总结出以下几个值得注意的问题:

- ❑ 每个表单页类要继承自 FormPage 类。
- ❑ 必须在构造方法中指定每一页的 ID 及其名称, 在以后的编程中, 就可以根据这个 ID 获得对该页的引用。
- ❑ 创建表单的内容区域是在 createFormContent 方法中实现的。
- ❑ 与在视图中创建 FormToolkit 对象不同, 在编辑器中获得 FormToolkit 对象是通过 IManagedForm 对象的 getToolkit()方法来获得的, 这是因为 FormToolkit 是占用系统资源的, 需要使用完释放。对于一个多页的编辑器来说, 可以共享一个 FormToolkit 对象, 这样只需要在释放编辑器时释放这个 FormToolkit 对象即可。而不必分别为每个页面都设置一个 FormToolkit 对象。
- ❑ 对于 SecondPage 类中的代码与 FirstPage 类中的代码类似, 这里就不多作解释了。

24.2.3 SWT 程序中使用表单

上文提到过，表单不仅可以应用在 Eclipse 插件开发和 RCP 开发中，也可以应用在任何 SWT/JFace 的 UI 界面的程序中。如图 24.4 所示，为一个简单的 SWT 窗口显示的表单效果。

该程序的具体实现代码如下：

NonEclipse.java

```
package com.fengmanfei.myrpc.forms;

import org.eclipse.swt.layout.FillLayout;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.ui.forms.widgets.FormToolkit;
import org.eclipse.ui.forms.widgets.ScrolledForm;

public class NonEclipse {

    public static void main(String[] args) {
        Display display = new Display();
        Shell shell = new Shell(display);
        shell.setText("SWT 表单示例");
        shell.setLayout( new FillLayout());
        //创建表单工具对象
        FormToolkit toolkit = new FormToolkit(shell.getDisplay());
        //通过表单工具对象创建可滚动的表单对象
        ScrolledForm form = toolkit.createScrolledForm(shell);
        //设置表单文本
        form.setText("Hello, Non Eclipse 表单");
        shell.pack();
        shell.open();
        while (!shell.isDisposed()) {
            if (!display.readAndDispatch())
                display.sleep();
        }
        toolkit.dispose();
        display.dispose();
    }
}
```



图 24.4 SWT 窗口中的表单

这就是 SWT 程序中最简单一个窗口。SWT 程序中应用了 FormToolkit 对象，就可以显示出表单的效果了，所做的只是导入表单所使用的类包即可。

24.2.4 获得表单工具对象 (FormToolkit) 一般方法

通过以上 3 种使用表单的方法可以看出, FormToolkit 对象在表单的使用中非常重要, 以下总结出创建或是获得 FormToolkit 的两种方法。

- ❑ 通过 Display 对象来创建, 例如以下代码:

```
FormToolkit toolkit = new FormToolkit(Display().getCurrent());
```

- ❑ 若在多页表单中不要使用 new 的方式, 而是要通过 IManagedForm 对象获得。例如以下代码:

```
FormToolkit toolkit=managedForm.getToolkit();
```

获得了 FormToolkit 就可以使用该对象的 createXXX 方法来创建各种所需的控件了。这些便是下文中所要讲述的内容。

24.3 表单的各种控件

通过前面的介绍, 读者已经从概念上了解什么是 Eclipse 表单, 知道表单可以为开发人员提供类似于 Web 网页的精美小控件。同时也清楚了基于 Eclipse 表单的开发所需要做的准备工作。下面就详细介绍一下主要的 Eclipse 表单的各种控件。

24.3.1 可滚动的表单 (ScrolledForm)

通过 SWT 的学习已经了解到, 若想放置一个基本的控件, 首先要有一个放置控件的容器, 不管是一个窗口 (Shell) 还是一个普通的面板 (Composite), 都可以作为容器来放置控件。对 Eclipse 表单来说, 也有对应的表单容器, 最基本的就是 ScrolledForm 对象。从图 24.5 所示的 ScrolledForm 类的继承关系图中可以清楚地看出, 该类继承自 Composite, 是一个容器类。

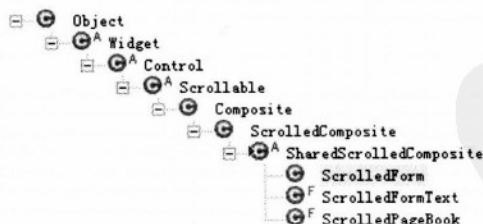


图 24.5 滚动的表单面板 ScrolledForm 类的继承关系图

可滚动的表单 (ScrolledForm) 是放置其他表单控件的容器, 可以实现带滚动的效果。与其他控件相比, ScrolledForm 比较特殊的是, 它也有两种方式获得:

- ❑ 如果是在 `FormPage` 中，可以利用 `createFormContent` 方法的 `IManagedForm` 对象获得。例如代码如下：

```
ScrolledForm form=managedForm.getForm();
```

- ❑ 如果是在其他地方，例如说在视图中使用，可以通过 `FormToolkit` 对象的 `createForm` 方法获得，代码如下：

```
ScrolledForm form = toolkit.createScrolledForm(parent);
```

另外，`ScrolledForm` 还可以设置标题栏、设置背景图片等使内容区可滚动的设置来仿照 Web 网页效果。例如，以下代码所示为设置了 `ScrolledForm` 对象的背景图片：

```
ScrolledForm form = toolkit.createScrolledForm(parent);
//设置表单文本
form.setText("Hello, Eclipse 表单");
//设置表单的背景图片
form.setBackgroundImage( MyRCPlugin.getImageDescriptor("icons/form_bg.gif").createImage());
```

- ❑ `setText` 方法可以设置表单的标题。
- ❑ `setBackgroundImage` 方法可以设置表单的背景图片。

24.3.2 可折叠的面板（ExpandableComposite）

在 Web 网页的 UI 体系中，最常见的也是效果最好的就是使页面中的部分区域中的内容可以折叠和展开，Eclipse 表单也提供了可折叠的面板（`ExpandableComposite`）来实现相同的功能。如图 24.6 和图 24.7 所示为一个放置了一个标签（`Label`）的可折叠的面板折叠和展开时的不同效果。

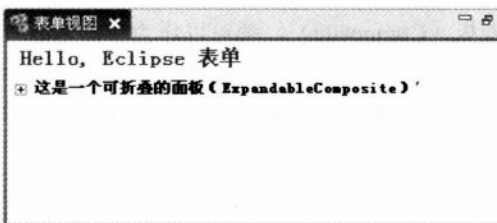


图 24.6 折叠时的效果

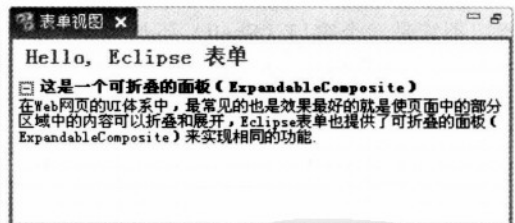


图 24.7 展开时的效果

为了实现以上的效果，需要在上文中所提到的视图类 `MyFormView` 的 `createPartControl` 方法中添加以下代码：

MyFormView.java

```
//设置表单的布局
form.getBody().setLayout( new TableWrapLayout());
//创建可折叠的面板
ExpandableComposite expcomp = toolkit.createExpandableComposite(form.getBody(),
ExpandableComposite.TREE_NODE);
```

```
//定义字符串
String txt = "在 Web 网页的 UI 体系中，最常见的也是效果最好的就是使页面中的部分区域中的内容
可以折叠和展开，Eclipse 表单也提供了可折叠的面板（ExpandableComposite）来实现相同的功能。";
//创建一个标签并显示字符串
Label client = toolkit.createLabel(expcomp, txt, SWT.WRAP);
//为折叠面板设置标题
expcomp.setText("这是一个可折叠的面板（ExpandableComposite）");
// 将 Label 作为折叠面板折叠区的内容
expcomp.setClient(client);
// 为折叠面板添加展开/折叠时的监听器
expcomp.addExpansionListener(new ExpansionAdapter() {
    public void expansionStateChanged(ExpansionEvent e) {
        // 根据部件的新尺寸重新定位和更新滚动条
        form.reflow(true);
    }
});
```

通过以上的代码，总结出创建可折叠面板时所需要注意的问题：

(1) 在表单中添加各种控件之前，需设置表单的布局，否则不会正确地显示控件。获得表单所在面板的方法是 `getBody`，然后就可以使用设置面板的方法设置布局了。例如，本例中设置的布局为 `TableWrapLayout` 布局，这种布局在之前还未使用过，它是专门为表格所设计的布局，布局的效果类似于网页中使用的表格布局，该布局的具体知识将在表单的高级应用部分详细讲述，读者在这里只需了解该表格是一种表单的布局就可以了。当然，表单除了可以使用 `TableWrapLayout` 布局外，也可以使用之前学习的 `FillLayout`、`GridLayout` 等布局，这要视不同的情况而定。

(2) 创建可折叠面板时，使用了 `FormToolkit` 对象以下所示的方法：

```
ExpandableComposite createExpandableComposite(Composite parent, int expansionStyle)
```

其中，`parent` 为创建该面板的父面板，本例中父面板为表单面板。`expansionStyle` 为可折叠面板的样式，可使用的样式常量如表 24.1 所示。

表 24.1 可折叠面板（`ExpandableComposite`）的样式常量及其效果示意图

样 式 常 量	说 明	展开时效果	折叠时效果
<code>ExpandableComposite.TREE_NODE</code>	展开按钮以树节点的样式	 可折叠的面板 这是折叠的区域	 可折叠的面板
<code>ExpandableComposite.TWISTIE</code>	展开按钮以三角形的样式	 可折叠的面板 这是折叠的区域	 可折叠的面板
<code>ExpandableComposite.CLIENT_INDENT</code>	内容区域缩进，与标题对齐，通常与 <code>TREE_NODE</code> 和 <code>TWISTIE</code> 样式连用，以“ ”隔开	 可折叠的面板 这是折叠的区域	

(3) 创建可折叠面板时，并不是直接创建使用 `ExpandableComposite` 的构造方法来创建可折叠面板对象，而是通过表单工具类 `FormToolkit` 对象中的 `createExpandableComposite`

方法来创建的,这样做更适合创建表单的控件。为了使读者更清楚使用工具类创建折叠面板的过程,以下显示了该方法的源代码,读者就更能体会到使用 `FormToolkit` 对象来创建面板的好处了。

```
public ExpandableComposite createExpandableComposite(Composite parent,
    int expansionStyle) {
    ExpandableComposite ec = new ExpandableComposite(parent, orientation, expansionStyle);
    ec.setMenu(parent.getMenu());
    adapt(ec, true, true);
    ec.setFont(boldFontHolder.getBoldFont(ec.getFont()));
    return ec;
}
```

从源代码中可以看出,通过 `FormToolkit` 对象来创建面板,可以不用管表单的菜单、字体等设置,该方法中已经是使用了默认的设置。这样就隐藏了表单底层一些实现,通过使用很简单的代码就可以创建出表单所需的控件。

这种通过 `FormToolkit` 表单创建控件的方法非常重要。因为在表单中,所有的控件都是通过 `FormToolkit` 对象的 `createXXX` 方法来创建,XXX 表示不同的控件。以下列举了在表单中创建之前学习 SWT 时一些常用控件的方法。

- ❑ `Button createButton(Composite parent, String text, int style)`: 创建按钮,其中 `text` 为按钮显示文字。
- ❑ `Composite createComposite(Composite parent, int style)`: 创建面板。
- ❑ `Label createLabel(Composite parent, String text, int style)`: 创建标签。
- ❑ `Label createSeparator(Composite parent, int style)`: 创建分隔符标签。
- ❑ `Table createTable(Composite parent, int style)`: 创建表格。
- ❑ `Text createText(Composite parent, String value, int style)`: 创建文本框。
- ❑ `Tree createTree(Composite parent, int style)`: 创建树。

这样,在表单中创建各种控件时,都需要使用这些方法来创建。

(4) 通过 `ExpandableComposited` 对象的 `setText` 方法可以设置折叠面板的标题。另外最重要的是设置折叠面板显示的控件是通过 `setTextClient(Control textClient)` 方法来设置的,其中 `textClient` 可以是任何的 `Control` 类及其子类。除了这最基本的方法外, `ExpandableComposited` 还有其他一些常用的方法。

- ❑ `setActiveToggleColor(Color c)`: 设置鼠标移到面板的标题上时标题的颜色。
- ❑ `setBackground(Color bg)`: 设置标题的背景色。
- ❑ `setForeground(Color fg)`: 设置标题的前景色。
- ❑ `setExpanded(boolean expanded)`: 设置折叠或是展开状态。
- ❑ `isExpanded()`: 获得当前折叠面板的状态, `true` 表示展开, `false` 表示折叠。

(5) 最后使用 `ExpandableComposited` 对象时一定要注册 `addExpansionListener` 监听器,这将会使展开和折叠面板式重新布局,否则将不能显示折叠和展开的效果。

24.3.3 内容区 (Section)

内容区 (Section) 是可折叠的面板 (ExpandableComposite) 子类, 如图 24.8 所示为类继承关系示意图。

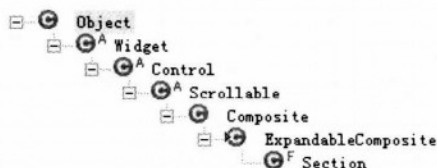


图 24.8 类继承关系示意图

除了具有 ExpandableComposite 类可折叠的特性外, 还可以设置不同的标题颜色、标题栏的分割线和标题栏的控件等。如图 24.9 和图 24.10 所示, 即为一个内容区展开和折叠时的效果示意图。

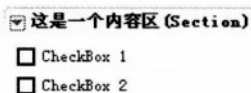


图 24.9 展开时的效果图

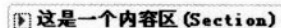


图 24.10 折叠时的效果图

实现这样效果的代码如下:

MyFormView.java

```

//创建内容区域, 使用 TWISTIE 样式, 并且使用 TITLE_BAR 显示背景的标题
Section section = toolkit.createSection(form.getBody(), Section.TWISTIE | Section.TITLE_BAR);
//设置标题文字
section.setText("这是一个内容区(Section)");
//创建一个面板, 作为内容折叠区放置的控件
Composite sectionClient = toolkit.createComposite(section);
sectionClient.setLayout( new GridLayout());
Button bt1 = toolkit.createButton(sectionClient, "CheckBox 1", SWT.CHECK);
Button bt2 = toolkit.createButton(sectionClient, "CheckBox 2", SWT.CHECK);
//设置内容区控件
section.setClient(sectionClient);
section.addExpansionListener(new ExpansionAdapter() {
    public void expansionStateChanged(ExpansionEvent e) {
        // 根据控件的新尺寸重新定位和更新滚动条
        form.reflow(true);
    }
});

```

通过以上代码可以看出创建内容区的方法需要注意以下几点:

- 创建 Section 对象是通过 FormToolkit 的 createSection 方法来创建的。

- ❑ 样式中若选用 `Section.TITLE_BAR` 则显示带背景颜色的标题, 否则与滚动面板的效果一样。
- ❑ 通过使用 `setClient` 方法, 可以为内容区添加所折叠显示的控件。
- ❑ 也可以设置内容区的描述信息, 例如以下代码:

```
//使用 Section.DESRIPTION 常量, 表示显示描述信息
Section section = toolkit.createSection(form.getBody(), Section.TWISTIE|Section.TITLE_BAR|
Section.DESRIPTION);
section.setDescription("这是在标题栏下的一段描述");//设置描述信息
```

添加了描述信息后, 内容区描述信息显示在标题的下方, 如图 24.11 所示。

- ❑ 可以设置内容区的分割线, 例如以下代码:

```
toolkit.createCompositeSeparator(section);//通过工具类来创建分割线
```

添加了分割线后的内容区的效果如图 24.12 所示。

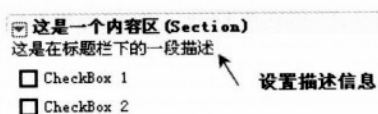


图 24.11 添加了描述信息的内容区效果图

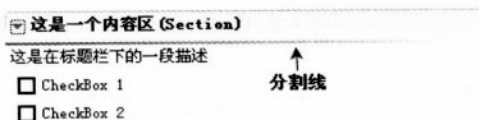


图 24.12 添加分割线后的内容区示意图

- ❑ 在 Eclipse 开发环境中的很多 `Section` 中, 都会有一个帮助按钮显示在 `Section` 的标题栏的右端, 非常美观且方便使用。如图 24.13 所示为一个 Eclipse 开发环境中使用的一个标题栏帮助按钮。

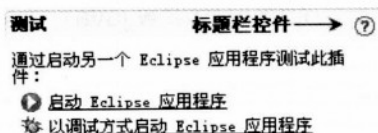


图 24.13 Eclipse 开发环境中带有标题栏控件的效果图

要添加这样的按钮, 只需要使用以下方法即可:

```
setTextClient(Control textClient)
```

其中, `textClient` 为任何一个 `Control` 类型的控件, 可以是按钮、工具栏、链接或者带图标的链接都可以显示在内容区标题栏的工具栏中。例如, 以下所示为内容区添加了一个图像超链接 (`ImageHyperlink`) 的代码。图像超链接也是一种表单的控件, 将在下文介绍。

```
//创建带图标的超链接
ImageHyperlink imageHyperlink = toolkit.createImageHyperlink(section, SWT.CENTER);
//设置超链接的图标
imageHyperlink.setImage( PlatformUI.getWorkbench().getSharedImages().getImageDescriptor(
SharedImages.IMG_TOOL_NEW_WIZARD).createImage());
//将该图标超链接添加到内容区的工具栏中
section.setTextClient( imageHyperlink );
```

该代码运行后,效果如图 24.14 所示。

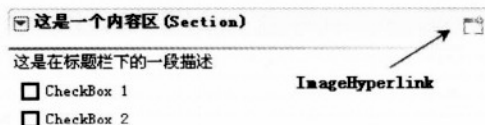


图 24.14 添加图像超链接的内容区效果示意图

24.3.4 超链接 (Hyperlink)

超链接 (Hyperlink) 是 Eclipse 表单中非常重要的一个控件,页面效果与在普通网页中看到的超链接非常类似,是一种带有下划线标记的标签,当获得鼠标焦点时可以变为手型,并且单击后可以响应鼠标单击事件,在网页中链接所具备的特征。如图 24.15 和图 24.16 所示为 Eclipse 开发环境中使用的两种不同类型的超链接示意图。



图 24.15 图标超链接

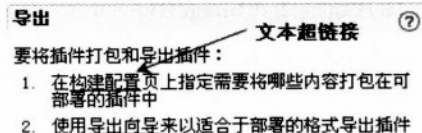


图 24.16 文本超链接

超链接有两种,一种是只有文本的超链接 (Hyperlink),一种是既可带图标又可带文本的超链接 (ImageHyperlink)。事实上,ImageHyperlink 是 Hyperlink 的子类,如图 24.17 所示为超链接所涉及的类继承关系示意图。

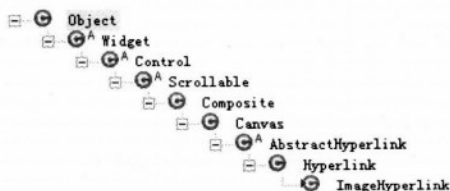


图 24.17 超链接类继承关系示意图

1. 文本超链接

文本超链接 (Hyperlink) 是最简单的超链接,可以使用 FormToolkit 以下所示的方法创建:

```
Hyperlink createHyperlink(Composite parent, String text, int style)
```

其中, parent 为链接所在的面板对象, text 为超链接显示的文本, style 为超链接的样式,可以为 SWT.WRAP, 表示可以折行显示文本。例如,以下代码为创建文本链接的代码:

```
//创建超链接,文字可以折行显示
Hyperlink textLink = toolkit.createHyperlink(linkComp,"这是一个文本超链接",SWT.WRAP);
//注册超链接事件监听器
textLink.addHyperlinkListener( new IHyperlinkListener(){
```

```

    public void linkEntered(HyperlinkEvent e) {
        //光标进入超链接区域
    }
    public void linkExited(HyperlinkEvent e) {
        //光标离开超链接区域
    }
    public void linkActivated(HyperlinkEvent e) {
        //单击超链接激活时
        System.out.println("超链接被激活");
    }
}
});

```

代码运行后，效果如图 24.18 所示。

其中，若要想处理超链接的事件，需要注册 `Ihyperlink Listener` 监听器。在 `linkActivated` 方法中处理单击链接所发生的事件。

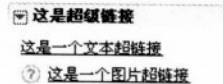


图 24.18 代码运行后效果示意图

2. 图片超链接 (ImageHyperlink)

带有图片的超链接 (`ImageHyperlink`) 除了可以设置文本外，还可以设置链接所显示的图片。创建带图片超链接的方法是：

```
ImageHyperlink createImageHyperlink(Composite parent, int style)
```

其中，`parent` 和 `style` 参数与 `Hyperlink` 的参数相同。例如，以下代码为创建带有图片的超链接的代码：

```

//创建带有图片的超链接
ImageHyperlink imageLink = toolkit.createImageHyperlink( linkComp ,SWT.WRAP );
//设置超链接的图标
imageLink.setImage( MyRCPPlugin.getImageDescriptor("icons/help_contents.gif").createImage());
//设置超链接的文本
imageLink.setText("这是一个图片超链接");
//设置当鼠标放到超链接上时的图标
imageLink.setHoverImage( MyRCPPlugin.getImageDescriptor("icons/linkto_help.gif").createImage());
//注册超链接监听器，使用 HyperlinkAdapter 适配器对象
imageLink.addHyperlinkListener( new HyperlinkAdapter(){
    public void linkActivated(HyperlinkEvent e) {
        System.out.println("超链接被激活");
    }
});

```

代码运行后，如图 24.18 中第二个超链接所示。

3. 超链接组 (HyperlinkGroup)

如果一个表单中有很多个超链接，此时就需要有一套管理机制来集中管理这些超链接，这就是超链接组 (`HyperlinkGroup`)。每个超链接通过 `add` 方法添加到 `HyperlinkGroup` 中，使用 `FormToolkit` 对象创建超链接时，已经自动将该超链接添加到超链接组中，此时通过

getHyperlinkGroup()方法可以获得该表单中超链接组。例如以下代码：

```
Hyperlink textLink = toolkit.createHyperlink(linkComp,"这是一个文本超链接",SWT.WRAP);//创建一个文本超链接
ImageHyperlink imageLink = toolkit.createImageHyperlink(linkComp,SWT.WRAP);//创建一个带图标的超链接
//获得超链接组
HyperlinkGroup group = toolkit.getHyperlinkGroup();
```

这样获得 HyperlinkGroup 对象后，就可以统一为所有的超链接进行设置和处理。

HyperlinkGroup 提供如下方法：

- ☐ setActiveBackground(Color newActiveBackground): 设置激活超链接时的背景色。
- ☐ setActiveForeground(Color newActiveForeground): 设置激活超链接时的前景色。
- ☐ setBackground(Color newBackground): 设置超链接的背景色。
- ☐ setForeground(Color newForeground): 设置超链接的前景色。
- ☐ setHyperlinkUnderlineMode(int mode): 设置超链接下划线的样式。
- ☐ getLastActivated(): 获得上一次所激活的超链接。

这样，通过 HyperlinkGroup 对象，就可以统一对链接组中的链接进行统一的设置和管理。

24.3.5 表单文本 (FormText)

以上的章节中了解到表单中提供了很多功能强大的控件：标签、文本、图片、超链接、图片链接等，使用这些控件可以轻松地创建各种类似“Web 网页中的效果”。但是，若要想构造复杂的控件，例如如图 24.19 所示的界面，按照之前学习使用的控件达到这样的效果，代码会很复杂，至少要多个标签和超链接控件才可以实现。

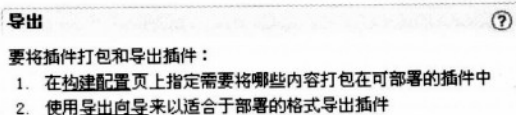


图 24.19 复杂的标签文本

这显然不符合编写程序时要尽量简化程序的原则，本来简单事情但实现却复杂。所以这里就产生了一个问题，表单既然倡导“Web 网页中的效果”，能不能按照 html 的形式来产生网页的效果呢？答案是肯定的，使用表单文本控件 (FormText) 就可以实现。

FormText 针对文本能够提供以下功能：

- ☐ 支持无格式文本的自动换行。
- ☐ 支持将无格式文本中任意一个以 http://开头的文本自动转换为超链接控件。
- ☐ 支持带有 XML 标记的文本。

1. 超链接转化

下面用一个例子来说明如何将表单文本中的以 http://开头的文本自动转化成超链接。如下代码为使用普通的文本转化成超链接的代码：


```

//创建一个表单本文对象, true 表示当该表单获得焦点时, 将超级链接的焦点设置为可见
FormText formText = toolkit.createFormText(form.getBody(), false);
//定义一个字符串, 其中包含 http://格式的字符
String text = "这是无格式的文本"+"这是带有超链接的文本 http://www.eclipse.org 将自动转化为超链接";
//设置文本 false 表示不转化 tag 标记, true 表示转化超链接
formText.setText(text, false, true);
//注册单击超链接监听器
formText.addHyperlinkListener(new HyperlinkAdapter(){
    //当单击链接时
    public void linkActivated(HyperlinkEvent e) {
        //打印出单击的超链接地址
        System.out.println("单击了该超链接: "+e.getHref());
    }
});

```

代码运行后的效果如图 24.20 所示。

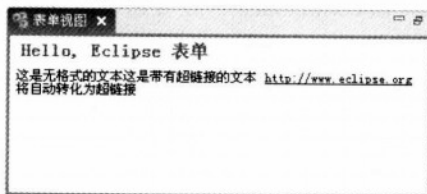


图 24.20 自动转化成超链接的文本

使用 FormText 转化超链接时主要注意以下几个问题:

- ☐ FormText 对象只将 http://及其以后的字符转化成超链接, 使用空格结束超链接。
- ☐ 使用以下所示的方法设置文本和转化的规则:

```
setText(String text, boolean parseTags, boolean expandURLs)
```

其中, text 为显示的字符串; parseTags 若为 true 表示将 xml 格式的标记转化成表单的格式, 该部分内容将在下文讲述; expandURLs 若为 true 则将 http://开头格式的文本自动转化为超链接, 否则将以普通的文本方式显示。

- ☐ 若要想判断不同的超链接, 可以为表单文本注册 Hyperlink 事件, 通过 e.getHref() 可以获得单击超链接的目标地址。

2. XML 标记

FormText 除了可以转化超链接外, 也可以将 XML 文件中的标记转化成超链接、图片和列表等。类似于简单的 html 的标记。例如, 有这样的一个 xml 文件: demo.xml, 该 xml 文件的具体代码如下:

demo.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<form>
    <p>这是一个段落, 使用"p"标记, 可以使用 <a href="action1">超链接</a> , 也可以

```

```

        <b>加粗</b> 文本. 还可以显示图片 <img href="image1"/> 功能很强大。
    </p>
    <li style="text" value="1.">这是一组文本列表, 使用"li"标记, 与段落一样 可以使用 <a
href="action1">超链接</a>, 也可以
        <b>加粗</b> 文本. 还可以显示图片 <img href="image1"/> 功能也很强大。
    </li>
    <li style="text" value="2.">这是第二个列表</li>
    <li style="text" value="3." bindent="20" indent="40">这是第三个列表</li>
    <li>这是一个无序列表, 默认为"bullet"样式</li>
    <li style="image" value="image1">这是带有图片的无序列表</li>
<p><span color="headColor" font="head">这是改变了文本的颜色字体的文本</span>
    </p>
</form>

```

将该 xml 文件中的标记转化为表单的格式后, 程序运行后的效果如图 24.21 所示。

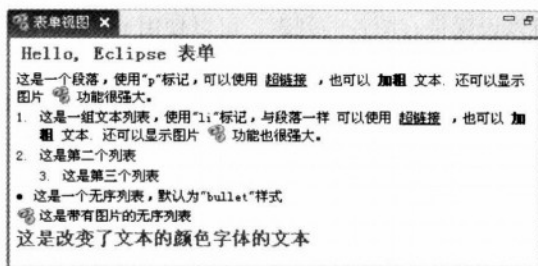


图 24.21 xml 文件转化成表单的效果示意图

最后, 要在 FormText 装载该 xml 文件还需要修改代码如下:

```

FormText formText = toolkit.createFormText(form.getBody(), false);
//设置文本所需要使用图片, 使用第一参数为图片的 key, 该值可被 xml 标记中的<img>标记中的 href
引用
formText.setImage("image1", MyRCPPlugin.getImageDescriptor("icons/samples.gif").createImage
());
//设置文本中所需要使用的字体, 第一个参数表示字体的 key, 值可被<span>的 font 属性引用
formText.setFont("head", form.getFont());
//设置文本中所需要使用的颜色, 第一个参数表示颜色的 key, 值可被<span>的 color 属性引用
formText.setColor("headColor", form.getForeground());
//装载与 MyFormView.java 同文件夹下的 demo.xml 文件
InputStream is = MyFormView.class.getResourceAsStream("demo.xml");
if (is!=null) { //若存在该文件
    //设置表单文本输入数据流
    formText.setContents(is, true);
    try {
        //最后关闭数据流
        is.close();
    }
    catch (IOException e) {
    }
}
}

```

下面来具体分析一下 xml 文件中每个标记所表示的意义。当使用文本中包含一些 FormText 所支持的标记时, 应该使用<form></form>作为根标记, 在根标记中, 就可以添加一个或多个其他常见的标记, 比如<p>或者。

□ <p>表示一个段落文本, 可用的属性有:

vspace: 如果设置为 false, 表示垂直方向上文本换行后将不留空隙, 默认为 true。例如以下代码:

```
<p vspace="false">这是一个段落</p>
```

□ 表示一个列表, 可表示有序的或无序的。可用的属性如下:

- ◆ vspace: 与<p>标记中的属性意义相同。
- ◆ style: 列表的类型, 若为无序的可设置为 bullet (默认值), 此时每个列表将以一个圆点显示。若设置为有序可以使用 text, 则配合 value 属性可以设置序号。也可以设置带有图片的列表, 可以使用 image, 同时配合 value 属性设置图片的 key 值。
- ◆ value: 配合 style 类型为 text 和 image 来使用, 对 bullet 不起作用。
- ◆ indent: 列表文本的缩进的像素值。
- ◆ bindent: 列表标号缩进的像素值。

在每个<p>和标记中又可以使用以下这些标记加粗、显示图片、显示超链接的效果:

□ img: 将转化为图片, 其中图片的地址使用 href 属性, 图片的地址为 setImage 方法中所设置的 key 值, 还可以设置属性 align, 值可以为 top、middle、bottom。例如以下代码:

```
<img href="image1" align="middle"/>
```

其中, image1 为 FormText 对象通过 setImage 方法设置的 key 值, 如下:

```
formText.setImage("image1", MyRCPPugin.getImageDescriptor("icons/samples.gif").createImage());
```

□ a: 将转化为超链接, 其中也使用 href 作为超链接的目标地址。例如以下代码:

```
<a href="action1">超链接</a>
```

□ b: 表示加粗文本显示, 例如以下代码:

```
<b>加粗</b>
```

□ br: 强制换行。

□ span: 可以设置一段文本的颜色和字体, 与标记的用法类似, 属性有 color 和 font, 值使用 setColor 和 setFont 代码中所设置的 key 值, 例如以下代码:

```
<span color="headColor" font="head">这是改变了文本的颜色字体的文本</span>
```

其中, headColor 为 FormText 对象通过 setColor 方法设置的 key 值, 如下:

```
formText.setColor("headColor", form.getForeground());
```

其中, head 为 FormText 对象通过 setFont 方法设置的 key 值, 如下:

```
formText.setFont("head",form.getFont());
```

- ❑ control: 此标记为 Eclipse 3.1 新增加的特性, 可以将 Control 控件转化为文本中的控件。与 img 标记用法相同, 通过 setControl 方法中设置的 key 值来获得控件的引用, 例如以下代码可以为文本中添加控件的标记:

```
<control href="myControl"></control>
```

其中, myControl 为 FormText 对象通过 setControl 方法设置的 key 值, 如下所示:

```
formText.setControl("myControl",toolkit.createButton(form.getBody(),"这是一个按钮",SWT.NONE));
```

3. 使用表单文本所注意的问题

虽然表单文本 (FormText) 已经提供了一些简单的转化的标记, 但使用时还有以下问题需要注意:

- ❑ 不能欠套标记, 例如在 标记中不能出现 标记, 虽然这在 html 中是允许的, 但在 FormText 中却不能这样使用。所以使用时要倍加小心。
- ❑ 在 Eclipse 3.1 以后的版本中, 可以将文本剪切、复制到剪贴板上, 但对于图片和控件却不能剪切或复制。
- ❑ 相对于复杂 html 标记而言, FormTex 的解析还比较简单。若使用复杂的 html 标记则需要使用 Browser 浏览器控件。

24.4 表单的布局管理器

在本书的第 7 章中, 已经介绍过 SWT 中常用的一些布局管理器。在使用表单时, 专门针对表单的特殊布局效果设计了两个新布局:

- ❑ 表格布局 (TableWrapLayout)。
- ❑ 列布局 (ColumnLayout)。

这两个布局都是基于 SWT 提供的布局。这些布局管理器实现了公共接口, 也可以用于 SWT 的面板容器中, 但通常与 UI 表单一起使用。

这些布局的目的是以类似于 Web 浏览器的方式管理表单布局。控件是根据表单宽度布置的。其目标是尽量保持表单的宽度并通过使表单垂直增大 (根据需要使用滚动条) 来弥补空间不足。

24.4.1 表格布局 (TableWrapLayout)

表格布局 (TableWrapLayout) 是基于网格的布局, 它与 SWT 的通用 GridLayout 非常类似。不同之处在于, 布局规则更类似于 HTML 中的布局算法, 可以支持文本的自动换行。

下面来比较一下 GridLayout 与 TableWrapLayout 两种布局的效果, 图 24.22 和图 24.23

所示为对同一个表单，分别使用了 GridLayout 和 TableWrapLayout 两种布局的不同效果示意图。

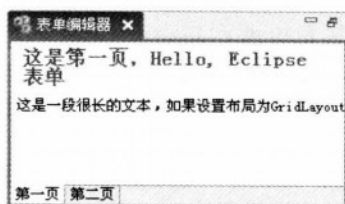


图 24.22 GridLayout 布局

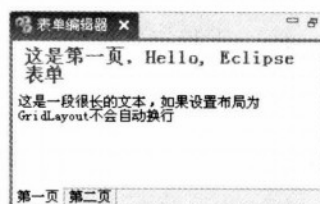


图 24.23 TableWrapLayout 布局

通过两种布局的比较可以看出，TableWrapLayout 可以将超出的文本折行显示，这是在 GridLayout 中所不能够实现的。

与 GridLayout 对应的 GridData 类似，TableWrapLayout 对应 TableWrapData。下面就以一个复杂的示例程序来说明如何使用 TableWrapLayout 进行布局设置。

该程序的代码如下：

FirstPage.java

```
package com.fengmanfei.myrpc.forms;

import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Label;
import org.eclipse.ui.forms.IManagedForm;
import org.eclipse.ui.forms.editor.FormEditor;
import org.eclipse.ui.forms.editor.FormPage;
import org.eclipse.ui.forms.widgets.FormToolkit;
import org.eclipse.ui.forms.widgets.ScrolledForm;
import org.eclipse.ui.forms.widgets.TableWrapData;
import org.eclipse.ui.forms.widgets.TableWrapLayout;

public class FirstPage extends FormPage {

    public static final String ID = "com.fengmanfei.myrpc.forms.FirstPage";

    public FirstPage(FormEditor editor) {
        // 构造方法，设置 Form 页的 ID 和名称
        super(editor, ID, "第一页");
    }

    // 覆盖父类中的方法
    // 在该方法中创建表单区域的各种控件
    protected void createFormContent(IManagedForm managedForm) {
        // 通过 managedForm 对象获得表单工具对象
        FormToolkit toolkit = managedForm.getToolkit();
        // 通过 managedForm 对象获得 ScrolledForm 可滚动的表单对象
        ScrolledForm form = managedForm.getForm();
    }
}
```



```

// 设置表单文本
form.setText("这是第一页, Hello, Eclipse 表单");
// 创建表格布局
TableWrapLayout layout = new TableWrapLayout();
layout.numColumns = 2;// 表格的列数
layout.bottomMargin = 10;// 下补白
layout.topMargin = 10;// 上补白
layout.leftMargin = 10;// 左补白
layout.rightMargin = 10;// 右补白
form.getBody().setLayout(layout);// 设置表格的布局

// 创建第一个标签
Label l1 = toolkit.createLabel(form.getBody(), "这是很长的一段文本文本 1", SWT.
WRAP);
// 创建第二个标签
Label l2 = toolkit.createLabel(form.getBody(), "这是文本 2", SWT.WRAP);
// 创建一个 TableWrapData 对象, 设置为水平和垂直充满式填充
TableWrapData td = new TableWrapData(TableWrapData.FILL_GRAB);
// 将布局数据应用到第二个标签
l2.setLayoutData(td);
Label l3 = toolkit.createLabel(form.getBody(), "这是文本 3", SWT.WRAP);
// 第三个标签的布局数据
td = new TableWrapData();
td.colspan = 2;// 设置单元格的跨两列
l3.setLayoutData(td);
// 第四个标签的布局数据
Label l4 = toolkit.createLabel(form.getBody(), "这是文本 4", SWT.WRAP);
td = new TableWrapData();
td.rowspan = 2;// 设置单元格跨两行
td.grabVertical = true;// 垂直抢占
l4.setLayoutData(td);
Label l5 = toolkit.createLabel(form.getBody(), "这是文本 5", SWT.WRAP);
Label l6 = toolkit.createLabel(form.getBody(), "这是文本 6", SWT.WRAP);

form.getBody().setBackground(form.getBody().getDisplay().getSystemColor(SWT.COLOR_
WIDGET_BACKGROUND));
}
}

```

该代码运行后效果如图 24.24 所示, 窗口改变后的效果如图 24.25 所示。

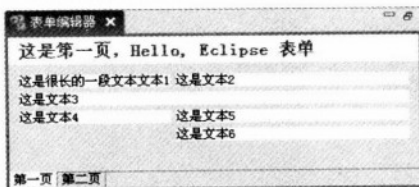


图 24.24 程序运行后的效果图



图 24.25 改变窗口大小后的效果

表格布局的使用方法和规则与 GridLayout 的使用方法非常类似，具体可以参考本书中 7.4 节中的内容。

24.4.2 列布局 (ColumnLayout)

列布局 (ColumnLayout) 与 SWT 中的 RowLayout 布局类似，相对于 TableWrapLayout 布局简单得多。该种布局的列数可根据窗口的大小进行调整，总是试图使用更多的列来显示，当窗口宽度减小时，列数也随之减少。总之，列布局 (ColumnLayout) 的目的是为了使各个控件都均匀地分布在界面上。

如图 24.26 和图 24.27 所示，为不同的窗口大小的情况下该布局的效果。

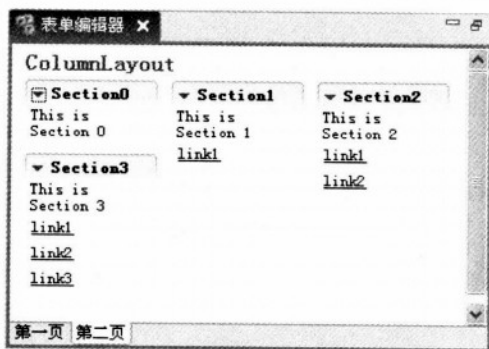


图 24.26 布局的原始状态

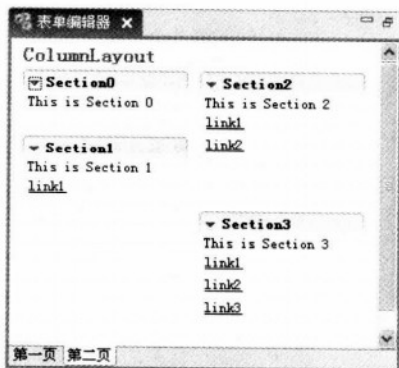


图 24.27 窗口大小调整后的状态

实现该布局的代码如下：

SecondPage.java

```
package com.fengmanfei.myrpc.forms;

import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.ui.forms.IManagedForm;
import org.eclipse.ui.forms.editor.FormEditor;
import org.eclipse.ui.forms.editor.FormPage;
import org.eclipse.ui.forms.events.ExpansionAdapter;
import org.eclipse.ui.forms.events.ExpansionEvent;
import org.eclipse.ui.forms.widgets.ColumnLayout;
import org.eclipse.ui.forms.widgets.FormToolkit;
import org.eclipse.ui.forms.widgets.ScrolledForm;
import org.eclipse.ui.forms.widgets.Section;

public class SecondPage extends FormPage {

    public static final String ID = "com.fengmanfei.myrpc.forms.SecondPage";
```

```

public SecondPage(FormEditor editor) {
    super(editor, ID, "第二页");
}

protected void createFormContent(IManagedForm managedForm) {
    ScrolledForm form = managedForm.getForm();
    form.setText("ColumnLayout");
    //创建列布局
    ColumnLayout layout = new ColumnLayout();
    layout.topMargin = 0;//上补白
    layout.bottomMargin = 5;//下补白
    layout.leftMargin = 10;//左补白
    layout.rightMargin = 10;//右补白
    layout.horizontalSpacing = 10;//水平方向控件的距离
    layout.verticalSpacing = 10;//垂直方向控件的距离
    layout.maxNumColumns = 4;//最大的列数
    layout.minNumColumns = 1;//最小的列数
    form.getBody().setLayout(layout);//设置表单的布局为列布局
    //创建 4 个内容区
    for (int i = 0; i < 4; i++)
        createSectionWithLinks(managedForm, "Section" + i, "This is Section " + i, i);
}
//创建内容区及其控件
private void createSectionWithLinks(IManagedForm mform, String title, String desc, int nlinks)
{
    //创建内容区面板
    Composite client = createSection(mform, title, desc, 1);
    FormToolkit toolkit = mform.getToolkit();
    //创建内容区中的控件
    for (int i = 1; i <= nlinks; i++)
        toolkit.createHyperlink(client, "link" + i, SWT.WRAP);
}
//创建内容区的方法
private Composite createSection(IManagedForm mform, String title, String desc, int num
Columns) {
    final ScrolledForm form = mform.getForm();
    FormToolkit toolkit = mform.getToolkit();
    //创建内容区
    Section section = toolkit.createSection(form.getBody(), Section.TWISTIE | Section.
TITLE_BAR | Section.DESRIPTION | Section.EXPANDED);
    section.setText(title);//设置标题
    section.setDescription(desc);//设置描述信息
    Composite client = toolkit.createComposite(section);
    GridLayout layout = new GridLayout();
    layout.marginWidth = layout.marginHeight = 0;
    layout.numColumns = numColumns;
    client.setLayout(layout);
    //设置内容区的面板

```

```
section.setClient(client);
section.addExpansionListener(new ExpansionAdapter() {
    public void expansionStateChanged(ExpansionEvent e) {
        form.reflow(false);
    }
});
return client;
}
```

24.5 表单的高级应用

表单的高级应用部分主要体现在提供了 Master/Details 模式的应用，通过表单可以轻松地实现这种模式。

24.5.1 Master/Details 模式

Master/Details 是 UI 设计中常见的一种模式。该种模式由一组（列表或成树状结构）Master 和一个被选中 Master 驱动的 Details 集组成。Master 是一些不同的对象，通过对 Master 的选择，驱动 Details 的 UI 发生变化。Master/Details 非常适用于 Eclipse 表单编程。

例如，编辑 plugin.xml 文件的扩展页面时就是使用的该种模式。如图 24.28 和图 24.29 所示，编辑扩展的左侧是所有的扩展列表，也就是 Master 部分。当单击左侧的不同的扩展信息时，右侧会产生不同的详细信息，也就是 Detail 部分。

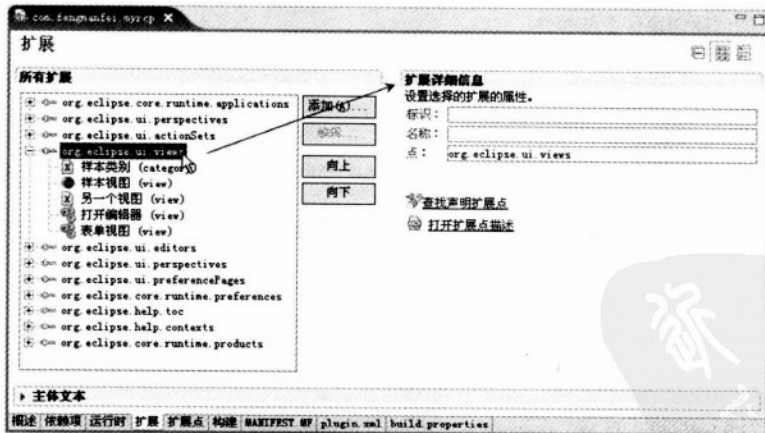


图 24.28 选中一个 Master，显示一个 Detail

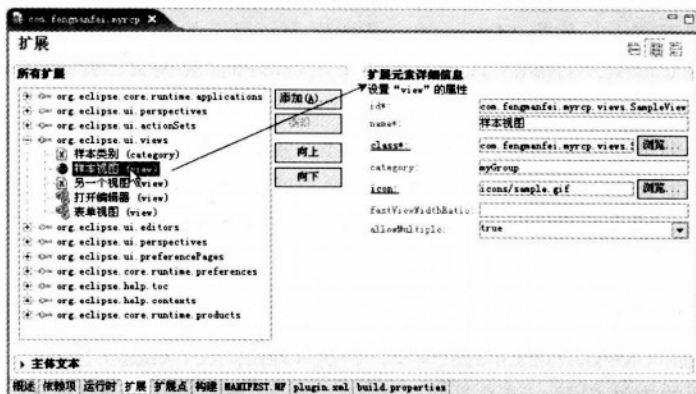


图 24.29 选中另一个 Master，显示另一个 Detail

24.5.2 实现 Master/Detail 示例程序

下面就以一个具体的例子来说明如何使用 Eclipse 表单来实现 Master/Detail 的界面效果。如图 24.30 和图 24.31 所示为本示例程序运行后的最终效果图。

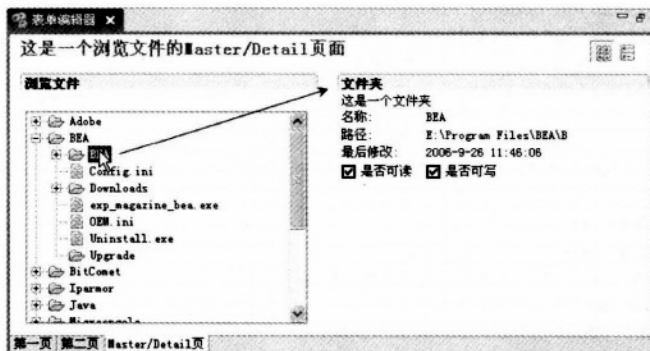


图 24.30 选中一个文件夹时，Detail 显示文件夹的详细信息

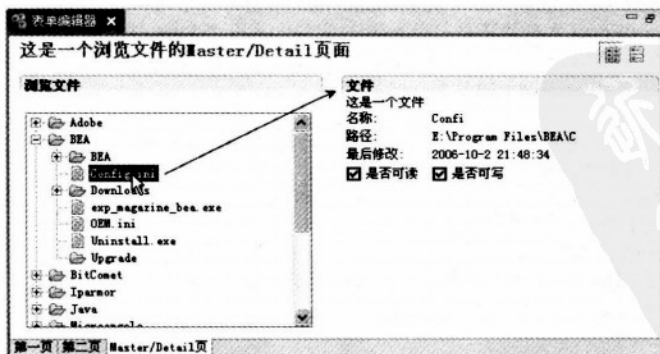


图 24.31 选中一个文件时，Detail 显示文件的详细信息

在这个示例程序中，作为 Master 的一个树界面，显示的是文件的结构目录，但左侧选中不同的文件或文件夹，右侧的 Detail 部分显示对应的详细信息。随着选中的不同，详细信息也不同。

下面就来详细地看一下如何实现这样的程序，步骤如下：

(1) 该页面为编辑器页面的一个页面，所以首先要在对应的编辑器页面添加一个页面。请读者参阅 24.2.2 小节中所提到的 MyMutiForm.java 类的源代码中，在 addPages 方法中添加一个页面，代码如下：

```
addPage(new MasterDetailPage(this));
```

(2) 所添加的 MasterDetailPage 对象，即为本例所示的 Master/Detail 页面。与之前所学习的创建页面的方法相同，该类是继承自 FormPage 类。以下为 MasterDetailPage 类具体实现的代码：

MasterDetailPage.java

```
package com.fengmanfei.myrpc.forms;

import org.eclipse.ui.forms.IManagedForm;
import org.eclipse.ui.forms.editor.FormEditor;
import org.eclipse.ui.forms.editor.FormPage;
import org.eclipse.ui.forms.widgets.ScrolledForm;
import com.fengmanfei.myrpc.forms.advance.FileMasterDetailsBlock;

public class MasterDetailPage extends FormPage {

    public static final String ID = "com.fengmanfei.myrpc.forms.MasterDetailPage";

    // 声明 MasterDetail 页面部分对象
    private FileMasterDetailsBlock block;
    public MasterDetailPage(FormEditor editor) {
        super(editor, ID, "Master/Detail 页");
        block = new FileMasterDetailsBlock(this);
    }

    /*
     * ManagedForm 封装了 form 元素的生命周期管理与各个 form 元素之间的事件通知
     * ManagedForm 本身并不是一个 form，它包含了一个 form 并且可以注册 IFormPart
     * 可以将 ManagedForm 看作是'viewers'，form 和 managed form 之间的关系就好像
     * Table 与 TableViewer 的关系一样
     */
    protected void createFormContent(IManagedForm managedForm) {
        // 获得表单对象
        ScrolledForm form = managedForm.getForm();
        // 设置表单的标题
        form.setText("这是一个浏览文件的 Master/Detail 页面");
        // 该方法非常重要，负责创建 Master 和 Detail 区域，尽量在最后调用
        block.createContent(managedForm);
    }
}
```

该页面与之前使用的编辑器页面不同，在构造方法中创建了一个 `FileMasterDetailsBlock` 对象，通过该对象的 `createContent` 方法可以创建 Master/Detail 页面中的各种界面，稍后将会详细讲述该类。

(3) 具体创建页面控件的部分是在 `FileMasterDetailsBlock` 类中实现的。该类继承自 `MasterDetailsBlock`，具有 Master/Details 模式的页面类都要继承自 `MasterDetailsBlock` 类，并实现该类的 3 个抽象方法。

- ❑ `createMasterPart`: 在该方法中创建 Master 部分的控件，可以是表格、树或列表等。本例中使用的是 `TreeView`。
- ❑ `createToolBarActions`: 创建页面的 Action 操作的方法，本例中创建两个按钮，可以设置垂直布局或是水平布局。
- ❑ `registerPages`: 注册 Master 对应的 Detail 部分的控件，可同时注册多个 Detail 页面。

实际上，一个 Master/Detail 页面是通过一个 `SashForm` 来布局的，Master 放置在 `SashForm` 的左侧或上侧，而 Detail 则放置在 `SashForm` 的右侧或下侧。

本例中创建界面的 `FileMasterDetailsBlock` 类的代码如下：

FileMasterDetailsBlock.java

```
package com.fengmanfei.myrpc.forms.advance;

import java.io.File;

import myRCP.MyRCPPlugin;

import org.eclipse.jface.action.Action;
import org.eclipse.jface.viewers.*;
import org.eclipse.swt.SWT;
import org.eclipse.swt.graphics.Image;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.widgets.*;
import org.eclipse.ui.*;
import org.eclipse.ui.forms.*;
import org.eclipse.ui.forms.editor.FormPage;
import org.eclipse.ui.forms.widgets.*;

public class FileMasterDetailsBlock extends MasterDetailsBlock {
    private FormPage page;

    public FileMasterDetailsBlock(FormPage page) {
        this.page = page;
    }
    //父类中的抽象方法，创建 Master 部分
    protected void createMasterPart(final IManagedForm managedForm, Composite parent) {
        FormToolkit toolkit = managedForm.getToolkit();
        //创建一个内容区
        Section section = toolkit.createSection(parent, Section.DESRIPTION | Section.
TITLE_BAR);
```

```

section.setText("浏览文件");
section.marginWidth = 10;
section.marginHeight = 5;
//创建内容区的面板
Composite client = toolkit.createComposite(section, SWT.WRAP);
//绘制该面板的边框, 与表单的风格一致
toolkit.paintBordersFor(client);
GridLayout layout = new GridLayout();
layout.numColumns = 2;
layout.marginWidth = 2;
layout.marginHeight = 2;
client.setLayout(layout);
//创建一个树, 使用 toolkit 对象创建
Tree tree = toolkit.createTree(client, SWT.NULL);
GridData gd = new GridData(GridData.FILL_BOTH);
gd.heightHint = 20;
gd.widthHint = 100;
tree.setLayoutData(gd);
/*

```

IFormPart 管理了整个 Part 的 dirty state, saving, commit, focus, selection changes 等这样的事件

并不是表单中的每一个空间站都需要成为一个 IFormPart, 最好将一组 control 通过实现 IFormPart 变成一个 Part.

一般来说 Section 就是一个自然形成的组, 所以 Eclipse Form 提供了一个 SectionPart 的实现, 它包含一个 Section 的对象

```

*/
final SectionPart spart = new SectionPart(section);
//注册该对象到 IManagedForm 表单管理器中
managedForm.addPart(spart);
//将普通的树包装成 MVC 的树
TreeViewer viewer = new TreeViewer(tree);
//注册树的选择事件监听器
viewer.addSelectionChangedListener(new ISelectionChangedListener() {
    //当单击树中某一个节点时
    public void selectionChanged(SelectionChangedEvent event) {
        //通过 IManagedForm 来发布 IFormPart 所对应的事件
        managedForm.fireSelectionChanged(spart, event.getSelection());
    }
});
//设置树的内容
viewer.setContentProvider(new MasterContentProvider());
//设置树的标签
viewer.setLabelProvider(new MasterLabelProvider());
//设置初始化输入的类
viewer.setInput(new File("E:\\Program Files"));
}
//注册详细页面部分
protected void registerPages(DetailsPart detailsPart) {
    //将 DirectoryDetailPage 对象注册

```

```

        detailsPart.registerPage(File.class, new DirectoryDetailPage());
    }
    //创建表单区的 Action
    protected void createToolBarActions(IManagedForm managedForm) {
        final ScrolledForm form = managedForm.getForm();
        //水平布局操作
        Action hAction = new Action("horizon", Action.AS_RADIO_BUTTON) {
            public void run() {
                sashForm.setOrientation(SWT.HORIZONTAL);
                form.reflow(true);
            }
        };
        hAction.setChecked(true);
        hAction.setToolTipText("水平布局");
        hAction.setImageDescriptor(MyRCPPlugin.getImageDescriptor("icons/hor.gif"));
        //垂直布局操作
        Action vAction = new Action("vertical", Action.AS_RADIO_BUTTON) {
            public void run() {
                sashForm.setOrientation(SWT.VERTICAL);
                form.reflow(true);
            }
        };
        vAction.setChecked(false);
        vAction.setToolTipText("垂直布局"); //$NON-NLS-1$
        vAction.setImageDescriptor(MyRCPPlugin.getImageDescriptor("icons/ver.gif"));
        //将这两个操作添加到表单的工具栏管理器中
        form.getToolBarManager().add(hAction);
        form.getToolBarManager().add(vAction);
    }

    public class MasterContentProvider implements ITreeContentProvider {
        .....代码省略
    }
    class MasterLabelProvider implements ILabelProvider {
        ..... 代码省略
    }
}

```

通过以上代码可以看出，创建的 Master 部分其实就是之前学习的浏览文件的 TreeViewer。但注意所不同的是，单击树中的某一个节点事件处理部分，需要通过 IManagedForm 的 fireSelectionChanged 方法来发布事件。

作为接收的事件处理，需要在注册的 Detail 页面进行，所注册的详细信息的部分都在 registerPages 方法中实现。本例中只注册了一个 Detail，如下：

```
detailsPart.registerPage(File.class, new DirectoryDetailPage());
```

当然也可以注册多个 Detail 页面。

(4) 最后看一下如何实现本例中的 Detail 页面，注册的 `DirectoryDetailPage` 对象即为具体实现 Detail 部分。该类需要实现 `IDetailsPage` 接口，主要是在 `createContents` 方法中创建 Detail 所需使用的控件，并且在 `selectionChanged` 方法中处理当 Master 对象选中事件发生的代码，其他的一些方法为接口中规定的一些方法，这里暂时空实现。该类的具体代码如下：

DirectoryDetailPage.java

```
package com.fengmanfei.myrpc.forms.advance;

import java.io.File;
import java.util.Date;

import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.viewers.IStructuredSelection;
import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.*;
import org.eclipse.ui.forms.*;

public class DirectoryDetailPage implements IDetailsPage {

    private IManagedForm mform;
    private File file;
    private Section fileSection;
    private Text fileName;
    private Text filePath;
    private Text lastModify;
    private Button isRead;
    private Button isWrite;
    private Composite client;

    public void createContents(Composite parent) {
        //设置父类面板的布局
        TableWrapLayout layout = new TableWrapLayout();
        layout.topMargin = 5;
        layout.leftMargin = 5;
        layout.rightMargin = 2;
        layout.bottomMargin = 2;
        parent.setLayout(layout);
        //获得表单工具对象
        FormToolkit toolkit = mform.getToolkit();
        //创建 Detail 的内容区
        fileSection = toolkit.createSection(parent, Section.DESRIPTION|Section.TITLE_BAR);
        TableWrapData td = new TableWrapData(TableWrapData.FILL, TableWrapData.TOP);
        td.grabHorizontal = true;
        fileSection.setLayoutData(td);
        //创建内容区所设置的面板
    }
}
```



```

        client = toolkit.createComposite(fileSection);
        fileSection.setClient( client );
        GridLayout gridLayout = new GridLayout();
        gridLayout.marginWidth = 0;
        gridLayout.marginHeight = 0;
        gridLayout.numColumns = 2;
        gridLayout.horizontalSpacing=10;
        client.setLayout(gridLayout);
        //创建 Detail 部分具体的各种控件
        toolkit.createLabel( client , "名称:");
        fileName = toolkit.createText( client , "");
        toolkit.createLabel( client , "路径:");
        filePath = toolkit.createText( client , "");
        toolkit.createLabel( client , "最后修改:");
        lastModify = toolkit.createText( client , file!=null?new Date(file.lastModified()). ToLocale
String():" ");
        isRead = toolkit.createButton( client , "是否可读" ,SWT.CHECK);
        isWrite = toolkit.createButton( client , "是否可写" ,SWT.CHECK);

    }
    //以下均为接口中的方法，这里暂时空实现
    public void initialize(IManagedForm form) {
        this.mform = form ;
    }
    public void dispose() {

    }
    public boolean isDirty() {
        return false;
    }
    public void commit(boolean onSave) {

    }
    public boolean setFormInput(Object input) {

        return false;
    }
    public void setFocus() {

    }
    public boolean isStale() {
        return false;
    }
    public void refresh() {

    }
    //当 Master 区域选中事件发生时
    public void selectionChanged(IFormPart part, ISelection selection) {
        //首先获得选中的对象

```

```
IStructuredSelection currentSelection = (IStructuredSelection)selection;
if (currentSelection.size() == 1)
    file = (File)currentSelection.getFirstElement();
//如果选中的对象不为 null, 则刷新控件所显示的值
if (file != null)
    update();
}
//刷新值
public void update () {
    //如果选中的为文件夹
    if (file.isDirectory()) {
        fileSection.setText("文件夹");
        fileSection.setDescription("这是一个文件夹");
    } else { //否则
        fileSection.setText("文件");
        fileSection.setDescription("这是一个文件");
    }
    //设置文件名
    fileName.setText(file.getName());
    //设置路径
    filePath.setText(file.getAbsolutePath());
    //设置上次修改
    lastModify.setText(new Date(file.lastModified()).toLocaleString());
    //设置是否只读
    isRead.setSelection( file.canRead());
    //设置是否可写
    isWrite.setSelection( file.canWrite() );
}
}
```

这样, 一个完整的 Master/Details 模式实现的表单就完成了。

24.6 本章小结

通过本章对 Eclipse 表单的学习, 读者对 SWT 控件的开发应该有了更深入的理解。通过使用表单的方法, 可以了解到 SWT 也可以自定义出不同样式的效果。

当然, Eclipse 表单的用途也很多, 除了用在编辑器中, 也可以应用在系统中。这要根据用户的喜好来决定。

第 25 章 项目实战——客户关系管理系统

本章将以一个具体的项目——客户关系管理系统为例，综合 SWT、JFace 和 RCP 的相关知识，学习如何进行实际项目的开发。使读者对项目的开发过程有一个整体的认识，能够快速将所学应用到实际的项目中。

25.1 系统概述

客户关系管理（Customer Relationship Management，CRM），是管理客户关系的系统，也是企业中很常用的应用系统。

25.1.1 系统预览

该系统是客户关系管理系统，主要目的是实现对客户各种信息的管理。在理解客户关系管理概念之前，首先来看一下本系统运行后主窗口界面最终的效果，如图 25.1 所示。

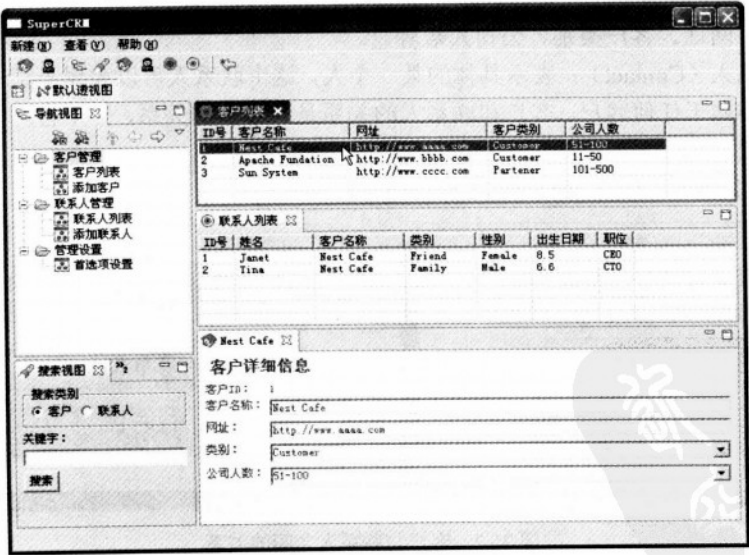


图 25.1 系统运行后的主界面的效果图

界面各个部分的说明具体如下：

- ❑ 导航视图：相当于导航菜单，用于打开所需要的操作。这些操作包括打开客户列表、

新建客户、打开联系人列表、新建联系人和打开首选项等。

- ❑ 客户列表视图：显示的是客户列表信息，包括根据查询条件进行搜索后的客户信息结果。
- ❑ 联系人列表视图：与客户列表视图类似，显示的是联系人的列表信息，包括根据查询条件进行搜索后的客户信息结果，并且当在客户列表中选中某一条客户信息后，在该视图中显示该客户下的所有联系人信息。
- ❑ 搜索视图：可以根据关键字搜索匹配的客户和联系人信息，并且搜索结果将显示在客户列表视图或联系人列表视图中。
- ❑ 客户详细信息编辑器：在客户列表视图中双击某一个客户信息，即打开该客户的详细信息编辑器，用户可以在这里修改客户的信息。

除了界面上所示的这些视图和编辑器界面外，还有以下未显示的视图的功能：

- ❑ 快速新建客户：在此视图中可以直接新建一个客户记录，新建后的结果显示在客户列表视图中。
- ❑ 快速新建联系人：在此视图中可以直接新建一个联系人记录，新建后的结果显示在联系人列表视图中。

25.1.2 基本概念介绍

客户关系管理系统在本系统中涉及两个基本的概念：客户和联系人。具体的说明如下：

- ❑ 客户（Customer）：可以表示一个公司、组织或机构，客户的基本信息包括客户名称、网址、客户类别、公司人数等。
- ❑ 联系人（Contact）：表示具体的某一个人，这个联系人可以隶属于某一个客户也可以不属于任何客户。客户和联系人的关系是一对多的关系，一个客户中可以包含多个联系人。

如图 25.2 所示为客户与联系人之间的关系。

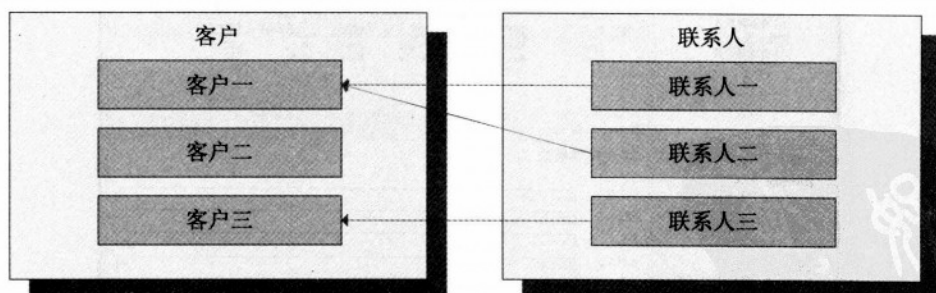


图 25.2 客户与联系人之间的关系

25.1.3 系统的运行环境

本系统需在以下环境中才可以运行：

- JDK 版本：JDK 1.5 以上。
- Eclipse 版本：Eclipse 3.1 以上。
- 数据库采用 MySQL：MySQL 5.0 版本以上。

25.1.4 系统文件结构的说明

在读者了解整个系统之前，首先对本系统中所涉及的类包的命名规则进行简要的说明。如图 25.3 所示为该项目中所涉及的类包。

各类包表示的意义如下：

- `superCRM.action`：包含了一些 Action 操作的类。
- `superCRM.business`：实现了业务逻辑的一些类。
- `superCRM.dao`：涉及数据库访问的类。
- `superCRM.dialog`：系统使用的一些对话框类。
- `superCRM.editor`：设计编辑器的一些类。
- `superCRM.intro`：主程序窗口的类。
- `superCRM.model`：设计业务模型的类。
- `superCRM.pojos`：数据库表对应的 POJO 类。
- `superCRM.preferences`：有关首选项所使用的类。
- `superCRM.table`：使用表格所涉及的类。
- `superCRM.util`：系统使用的工具类。
- `superCRM.views`：系统所涉及的视图类。

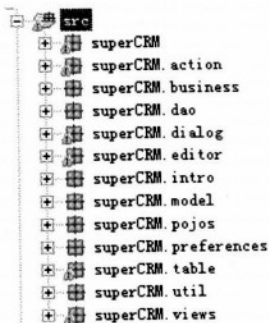


图 25.3 项目中所涉及的类包

25.2 UI 界面设计

了解了有关客户关系管理的基本概念，下面就介绍该系统各个层面的设计考虑，包括 UI 界面的实现、业务层和数据库层的设计等。

本系统 UI 部分采用 RCP 的开发方式。程序窗口的各个部分主要是由各种视图和编辑器组成的。如图 25.4 所示为主窗口的布局的设计图。

布局具体实现是在默认的透视图中所定义的。在本系统中，默认的透视图在 `plugin.xml` 文件中的定义代码如下：

plugin.xml

```

<extension
    point="org.eclipse.ui.perspectives">
    <perspective
        class="superCRM.intro.Perspective"
        icon="icons/logo.gif"
        id="SuperCRM.perspective"
        name="默认透视图">
    </perspective>
</extension>

```

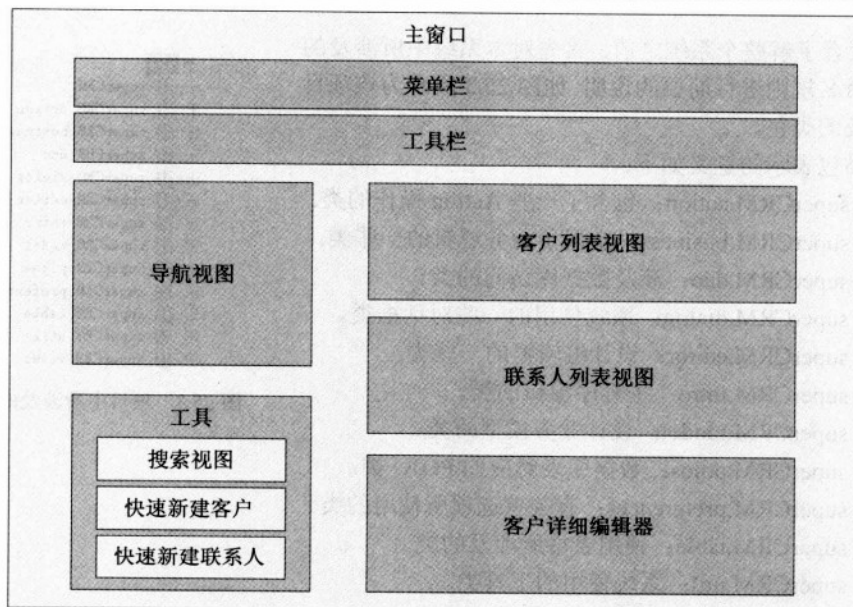


图 25.4 UI 布局设计图

配置所对应的透视图类为 `superCRM.intro.Perspective`，该类具体实现的代码如下：

Perspective.java

```

package superCRM.intro;

import org.eclipse.ui.IFolderLayout;
import org.eclipse.ui.IPageLayout;
import org.eclipse.ui.IPerspectiveFactory;
import superCRM.views.ContactSummaryView;
import superCRM.views.CustomerSummaryView;
import superCRM.views.NavView;
import superCRM.views.QuickNewContactView;
import superCRM.views.QuickNewCustomerView;
import superCRM.views.SearchView;

```

```
/** 默认的透视图类 */
public class Perspective implements IPerspectiveFactory {

    public void createInitialLayout(IPageLayout layout) {
        String editorArea = layout.getEditorArea();

        /** 编辑器的左侧放置导航视图 */
        layout.addView(NavView.ID, IPageLayout.LEFT, 0.25f, editorArea);

        /** 导航视图的下方放置一些工具视图，以文件夹的形式 */
        IFolderLayout leftBottom = layout.createFolder("left", IPageLayout.BOTTOM, 0.6f,
NavView.ID);

        /** 搜索视图 */
        leftBottom.addView(SearchView.ID);

        /** 快速新建联系人视图 */
        leftBottom.addView(QuickNewContactView.ID);

        /** 快速新建客户视图 */
        leftBottom.addView(QuickNewCustomerView.ID);

        /** 编辑器的上方放置客户列表视图 */
        layout.addView(CustomerSummaryView.ID, IPageLayout.TOP, 0.25f, editorArea);

        /** 编辑器的上方预留放置联系人列表视图 */
        layout.addPlaceholder(ContactSummaryView.ID, IPageLayout.TOP, 0.25f, editorArea);
    }
}
```

25.3 业务层设计

25.3.1 业务层服务的定义

在本系统中主要涉及两种业务：一个是客户的管理，定义为 `ICustomerService`；另一个是联系人的管理，定义为 `IContactService`。从设计结构的角度，这些业务都需要以接口的形式进行定义。

客户管理 `ICustomerService` 的接口中定义的代码如下：

`ICustomerService.java`

```
package superCRM.model;

import java.util.List;
```

```
import superCRM.pojos.CustomerEO;

/**有关对客户管理的服务*/
public interface ICustomerService {

    /**添加一个客户*/
    public CustomerEO addCustomer ( CustomerEO customer );

    /**更新客户信息*/
    public void updateCustomer ( CustomerEO customer );

    /**根据客户 ID 获得客户信息*/
    public CustomerEO getCustomer ( int id );

    /**根据关键字获得查询出的客户信息*/
    public List getCustomers ( String keywords );

    /**获得所有客户信息*/
    public List getAllCustomers ();

    /**获得某一个客户下的联系人*/
    public List getContacts ( CustomerEO customer );
}
```

联系人管理 IContactService 的接口中定义的代码如下：

IContactService.java

```
package superCRM.model;

import java.util.List;
import superCRM.pojos.ContactEO;

/**有关对联系人管理的服务*/
public interface IContactService {

    /**添加一个联系人*/
    public ContactEO addContact ( ContactEO contact );

    /**根据 ID 号获得该联系人的信息*/
    public ContactEO getContact ( int id );

    /**根据关键字获得查询结构的联系人*/
    public List getContacts ( String keywords );

    /**获得所有的联系人*/
    public List getAllContacts ();
}
```

25.3.2 业务层的实现

设计了业务层接口，最终还要看如何来实现这些业务逻辑的接口。首先来看一下实现 ICustomerService 接口类的具体代码，如下：

ICustomerService.java

```
package superCRM.business;

import java.util.List;
import superCRM.dao.DaoFactory;
import superCRM.dao.ICustomerDao;
import superCRM.model.ICustomerService;
import superCRM.pojos.CustomerEO;

public class CustomerService implements ICustomerService {

    /** 数据库访问的 DAO，通过这个对象来实现对数据库的操作 */
    private ICustomerDao customerDao;

    /** 构造方法 */
    public CustomerService() {

        /** 创建一个 DaoFactory 工厂对象 */
        DaoFactory daoFactory = new DaoFactory();
        /** 通过工厂方法来创建一个 ICustomerDao 对象 */
        customerDao = daoFactory.getCustomerDao();
    }

    public CustomerEO addCustomer(CustomerEO customer) {
        /** 添加一个客户 */
        return customerDao.addCustomer(customer);
    }

    public CustomerEO getCustomer(int id) {
        /** 查找指定客户 */
        return customerDao.findById(id);
    }

    public List getCustomers(String keywords) {

        /** 如果关键字为空或者为 null，则返回所有的客户 */
        if (keywords == null || keywords.equals(""))
            return getAllCustomers();
        /** 根据关键字获得查询客户结果 */
        return customerDao.findByKeywords(keywords);
    }
}
```



```
public List getAllCustomers() {  
    /** 获得所有的客户 */  
    return customerDao.getAllCustomers();  
}  
  
public void updateCustomer(CustomerEO customer) {  
    /** 更新客户 */  
    customerDao.updateCustomer(customer);  
}  
  
public List getContacts(CustomerEO customer) {  
    /** 获得该客户所属的联系人 */  
    return customerDao.getContacts(customer);  
}  
}
```

通过以上代码可以看出，在业务逻辑实现的过程中，涉及的数据库操作是通过 `ICustomerDao` 对象来实现的。这样就实现了数据库与业务逻辑的分离，当改变数据库时，并不需要改变业务逻辑的代码。25.4 节中还将对数据库的设计进行详细的讲述。

对于实现 `IContactService` 接口的类 `ContactService`，与 `CustomerService` 实现的代码类似，这里就不详细给出具体的代码了，请读者参阅光盘的源代码部分。

25.3.3 业务层服务的管理

当然，为了能够集中管理系统整个服务，还需要设计一个接口来管理客户服务和联系人服务。定义整个系统的各个服务的管理，这里使用 `ISuperApplication` 接口来定义管理整个系统的各个服务模块。该接口具体实现的代码如下：

ISuperApplication.java

```
package superCRM.model;  
  
/**定义整个系统的各个服务模块*/  
public interface ISuperApplication {  
  
    /**客户管理服务模块*/  
    public ICustomerService getCustomerService();  
  
    /**联系人管理服务模块*/  
    public IContactService getContactService();  
}
```

通过这样设计的好处是，当系统在后期进行维护时非常容易。例如，此时需要增加一个 Email 管理的功能，只需要在该接口中添加一个 `IEmailService()` 的方法即可。增加 Email

服务后代码如下：

```
public interface ISuperApplication {  
    public ICustomerService getCustomerService();  
    public IContactService getContactService();  
    /**添加的邮件服务*/  
    public IEmailService getEmailService ();  
}
```

以下是实现该接口类中的代码：

RcpApplication.java

```
package superCRM.business;  
  
import superCRM.model.IContactService;  
import superCRM.model.ICustomerService;  
import superCRM.model.ISuperApplication;  
  
public class RcpApplication implements ISuperApplication {  
  
    /** 定义客户服务对象 */  
    private ICustomerService customerService = null;  
  
    /** 定义联系人服务对象 */  
    private IContactService contactService = null;  
  
    /** 获得客户服务对象 */  
    public ICustomerService getCustomerService() {  
        if (customerService == null) {  
            customerService = new CustomerService();  
        }  
        return customerService;  
    }  
  
    /** 获得联系人服务对象 */  
    public IContactService getContactService() {  
        if (contactService == null) {  
            contactService = new ContactService();  
        }  
        return contactService;  
    }  
}
```

25.3.4 业务层 UML 图

为了更好地理解业务层各个类之间的关系，如图 25.5 所示为这些接口和类的关系 UML 图。

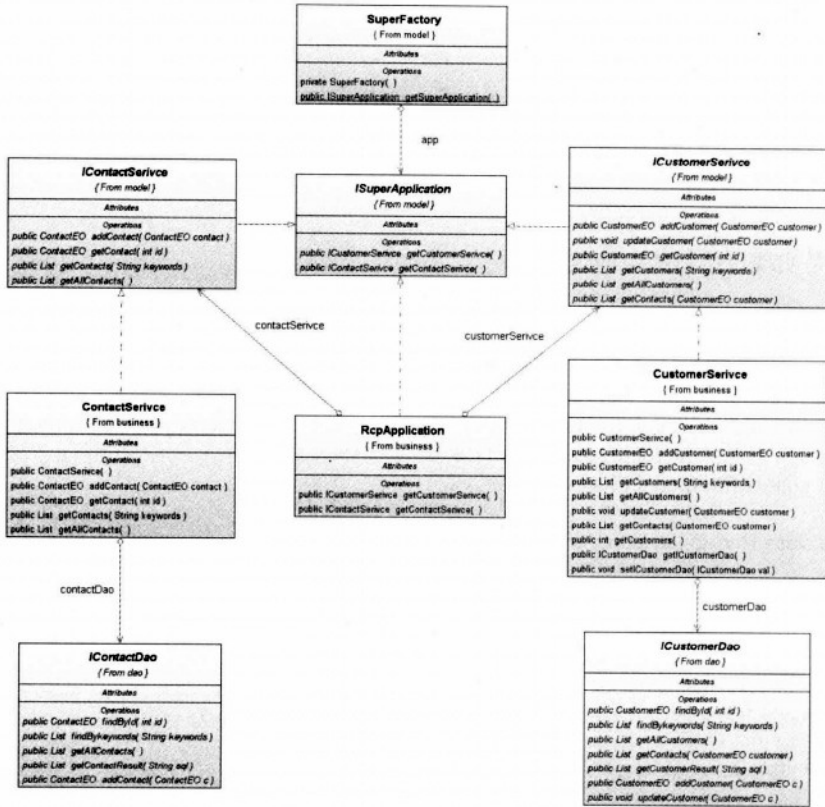


图 25.5 业务层 UML 类图

25.3.5 如何调用业务对象

理解了整个业务上的设计，那么如何创建对象，从而获得业务对象的实例呢？本系统采用了单例模式（Singleton）。具体实现的代码如下：

SuperFactory.java

```

package superCRM.model;

import superCRM.business.RcpApplication;

/**通过单例模式来创建管理整个系统的对象*/
public class SuperFactory {

    /**设置为私有类型，不允许外部类实例化*/
    private SuperFactory(){ }

    /**定义为静态类型*/
    private static ISuperApplication app = null ;

```

```

/**获得该实例*/
public static ISuperApplication getSuperApplication (){
    if ( app == null){
        app = new RcpApplication();
    }
    return app;
}
}

```

这样，通过该类就可以获得所需要的服务了。例如，以下所示为获得客户服务的代码：

```

ICustomerService customerService =
    SuperFactory.getSuperApplication().getCustomerService();

```

以下所示为获得联系人服务的代码：

```

IContactService contactService =
    SuperFactory.getSuperApplication().getContactService();

```

25.4 数据库层设计

下面来看一下数据库访问层的设计。对于支持多种数据库系统，要进行合理的设计才可以实现的。如图 25.6 所示为数据库层所涉及类的 UML 图。

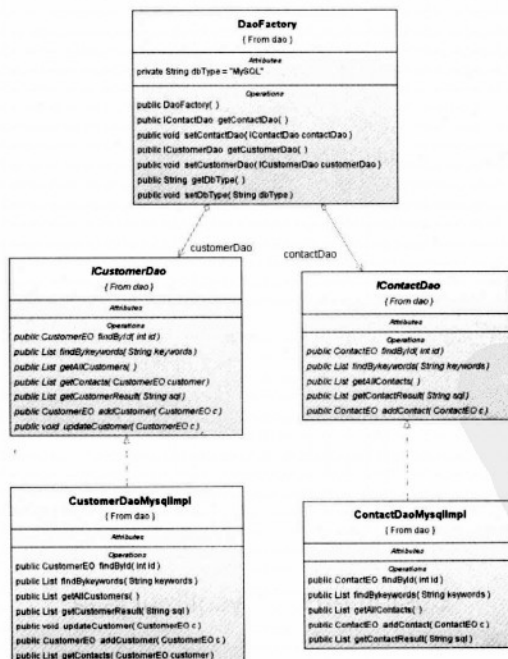


图 25.6 数据库层的 UML 图

25.4.1 数据库接口类

为了使系统适应不同的数据库，需要为每个服务定义一个数据库访问的 DAO 接口，在业务层中已经涉及了这些 DAO 接口类，这里只给出 ICustomerDao 接口的代码，而系统涉及的 IContactDao 与该接口中的代码类似。ICustomerDao 接口中的代码如下：

ICustomerDao.java

```
package superCRM.dao;

import java.util.List;
import superCRM.pojos.CustomerEO;

/** 客户数据库访问 DAO */
public interface ICustomerDao {

    /** 客户数据库访问 DAO */
    public CustomerEO findById(int id);

    /** 根据关键字查找获得查询结果 */
    public List findBykeywords(String keywords);

    /** 获得所有的客户记录 */
    public List getAllCustomers();

    /** 获得该客户的联系人 */
    public List getContacts(CustomerEO customer);

    /** 根据 SQL 语句获得客户记录 */
    public List getCustomerResult(String sql);

    /** 添加客户 */
    public CustomerEO addCustomer(CustomerEO c);

    /** 更新客户 */
    public void updateCustomer(CustomerEO c);
}
```

25.4.2 实现了 MySQL 数据库类

本系统中使用的是 MySQL 数据库，以下是实现了 ICustomerDao 接口的 MySQL 数据库的代码：

CustomerDaoMysqlImpl.java

```
package superCRM.dao;
```

```
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
import superCRM.pojos.CustomerEO;

public class CustomerDaoMysqlImpl implements ICustomerDao{

    public CustomerEO findById(int id) {
        int row = -1;
        // 读记录的 SQL 语句
        String sql = "select * from customer where customer_id=" + id
            + " and active_status='Y'";
        ResultSet rs = DbManager.getResultSet(sql); // 执行 SQL 语句并返回 ResultSet
        try {
            rs.last(); // 移动到最后一行
            row = rs.getRow(); // 得到总记录数
            if (row == 1) { // 如果只查询到一条记录, 则代表该记录存在并更新该类的属性
                CustomerEO customer = new CustomerEO();
                customer.setId( rs.getInt("customer_id"));
                customer.setDisplayName( rs.getString("display_name"));
                customer.setWebSite(rs.getString("web_site"));
                customer.setCategory(rs.getString("customer_category"));
                customer.setNumberEmployee(rs.getString("num_employee"));
                return customer;
            } else
                return null;
        } catch (SQLException e) {
            e.printStackTrace();
            return null;
        } finally {
            try {
                if (rs != null)
                    rs.close();
                if (rs.getStatement() != null)
                    rs.getStatement().close();
                DbManager.releaseConnection();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }

    public List findBykeywords(String keywords) {
        // 查询记录的 SQL 语句
        String sql = "select customer_id from customer where active_status='Y' "
            + "and (display_name like '%" + keywords + "%' or web_site like '%" + keywords + "%' or "
            + "customer_category like '%" + keywords + "%',)";
        return getCustomerResult(sql);
    }
}
```



```
}

public List getAllCustomers() {
    //查询所有客户记录的 SQL 语句
    String sql = "select customer_id from customer where active_status='Y' ";
    return getCustomerResult(sql);
}

public List getCustomerResult(String sql) {
    //执行 SQL 语句并返回 ResultSet
    ResultSet rs = DbManager.getResultSet(sql);
    List list = new ArrayList();
    try {
        while(rs.next()){
            int id = rs.getInt("customer_id");
            CustomerEO c = findById( id );
            list.add(c);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    DbManager.releaseConnection();
    return list;
}

public void updateCustomer(CustomerEO c) {
    //更新客户的 SQL 语句
    String sql = "update customer set display_name=' "+c.getDisplayName()+" ',web_site=
        ' "+c.getWebSite()+" ',
        customer_category=' "+c.getCategory()+" ',num_employee=' "+c.getNumberEmployee()
        +' ' where customer_id="+c.getId();
    try {
        DbManager.excute(sql);
    } catch (RuntimeException e) {
        e.printStackTrace();
    }
    DbManager.releaseConnection();
}

public CustomerEO addCustomer(CustomerEO c) {
    //添加客户的 SQL 语句
    String sql = "select customer_id from customer ";
    int row = getCustomerResult(sql).size()+1;
    sql = "insert into customer values (" + row + ", ' " + c.getDisplayName() + " ' +
        " ' " + c.getWebSite() + " ' , ' " + c.getCategory() + " ' , ' " + c.getNumberEmployee() +
        " ' , ' " + c.getActiveStatus() + " ' )";
    try {
        DbManager.excute(sql);
        DbManager.releaseConnection();
    }
```

```

        return findByld( row );
    } catch (RuntimeException e) {
        e.printStackTrace();
        return null;
    }
}

public List getContacts(CustomerEO customer) {
    //查询该客户的联系人 SQL 语句
    String sql = "select contact_id from contact where active_status='Y'and customer_
id="+customer.getId();
    DaoFactory daoFactory = new DaoFactory();
    return daoFactory.getContactDao().getContactResult(sql);
}
}

```

该类中对数据的查询和插入操作又是通过 DbManager 来实现的。该类的代码如下：

DbManager.java

```

package superCRM.dao;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class DbManager {
    //类成员 Connection
    protected static Connection conn;
    //mysql 的驱动类，定义为常量
    public static final String CLASS_NAME = "com.mysql.jdbc.Driver";
    //数据库的连接地址，定义为常量
    public static final String CONNET_STR = "jdbc:mysql://localhost/superCRM";
    //获得 Connection
    public static Connection getConnection() {
        try {
            Class.forName(CLASS_NAME); //使用类反射加载该驱动类
            //获得一个 Connection
            conn = DriverManager.getConnection(CONNET_STR, "root", "");
            return conn; //返回该 Connection
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
    //传入查询数据库的 SQL 语句，返回 ResultSet
    public static ResultSet getResultSet(String sql) {

```

```
boolean bSuccess = true;
Statement stmt = null;//声明 Statement stmt
ResultSet rs = null;//声明 ResultSet rs
Connection con = getConnection();//调用 getConnection()方法获得一个 Connection
if (con == null)//如果 Connection 为 null, 则返回假
    bSuccess = false;
if (bSuccess) {
    try {
        stmt = con.createStatement();//通过 Connection 创建一个 Statement
        rs = stmt.executeQuery(sql);//执行查询语句
    } catch (SQLException e) {
        e.printStackTrace();
        bSuccess = false;
    }
}
if (bSuccess)//如果执行成功, 则返回 rs
    return rs;
else
    return null;
}
//传入执行数据更新的语句, 返回更新结果, 真为成功执行
public static boolean excute(String sql) {
    boolean bSuccess = true;
    Statement stmt = null;//声明 Statement stmt
    Connection con = getConnection();//调用 getConnection()方法获得一个 Connection
    if (con == null)//如果 Connection 为 null, 则返回假
        bSuccess = false;
    if (bSuccess) {
        try {
            stmt = con.createStatement();//通过 Connection 创建一个 Statement
            bSuccess = stmt.executeUpdate(sql);//执行更新数据操作
        } catch (SQLException e) {
            e.printStackTrace();
            bSuccess = false;
        }
    }
    return bSuccess;
}
//释放 Connection
public static void releaseConnection() {
    try {
        if (conn != null)// 如果 Connection 不为 null, 则关闭 Connection
            conn.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
}
```

至此，一个完整的 MySQL 的数据库实现已经完成了。

25.4.3 如何调用数据访问对象

理解了 MySQL 数据库的实现，下面来看一下如何在业务中调用数据访问对象。在 CustomerService 类的代码中，可以发现调用数据访问对象是通过 DaoFactory 来获得的，代码如下：

```
public class CustomerService implements ICustomerService {
    private ICustomerDao customerDao;
    public CustomerService() {
        /** 创建一个 DaoFactory 工厂对象 */
        DaoFactory daoFactory = new DaoFactory();
        /** 通过工厂方法来创建一个 ICustomerDao 对象 */
        customerDao = daoFactory.getCustomerDao();
    }
}
```

DaoFactory 对象是如何获得 MySQL 数据访问对象的呢？首先来看一下 DaoFactory 类具体的代码，如下：

DaoFactory.java

```
package superCRM.dao;

public class DaoFactory {

    /** 定义数据库类型的常量 */
    public static final String MYSQL = "mysql";

    public static final String ORACLE = "oracle";

    public static final String SQLSERVER = "sql_server";

    /** 数据库的类型 */
    private String dbType = MYSQL;

    private ICustomerDao customerDao;

    private IContactDao contactDao;

    /** 构造方法 */
    public DaoFactory() {
        /** 如果数据库类型为 MySQL */
        if (dbType.equals(MYSQL)) {
            customerDao = new CustomerDaoMysqlImpl();
            contactDao = new ContactDaoMysqlImpl();
        }
    }
}
```



```
}

/** 一些 getter 和 setter 方法 */
public IContactDao getContactDao() {
    return contactDao;
}

public void setContactDao(IContactDao contactDao) {
    this.contactDao = contactDao;
}

public ICustomerDao getCustomerDao() {
    return customerDao;
}

public void setCustomerDao(ICustomerDao customerDao) {
    this.customerDao = customerDao;
}

public String getDbType() {
    return dbType;
}

public void setDbType(String dbType) {
    this.dbType = dbType;
}
}
```

因为该类默认的数据类型为 MySQL, 所以直接使用以下代码获得的是默认的数据实现, 但若要获得其他类型数据库实现的数据访问对象, 还需要对数据库的类型进行设置。

```
DaoFactory daoFactory = new DaoFactory();
customerDao = daoFactory.getCustomerDao();
```

25.4.4 应用多种数据库

设计数据库访问的 DAO 接口后, 当为系统增加新的数据库类型时, 实现该接口就可以了。例如, 增加了 SQL Server 的支持, 代码如下:

```
public class CustomerDaoSqlServerImpl implements ICustomerDao{
    //代码省略
}
```

例如, 增加了 Oracle 数据库时, 代码如下:

```
public class CustomerDaoOracleImpl implements ICustomerDao{
    //代码省略
}
```


然后再修改 DaoFactory 的构造方法中的代码如下：

```
public DaoFactory() {
    /** 如果数据库类型为 MySQL */
    if (dbType.equals(MYSQL)) {
        customerDao = new CustomerDaoMysqlImpl();
        contactDao = new ContactDaoMysqlImpl();
    } else if (dbType.equals(ORACLE)) { /** 如果数据库类型为 Oracle */
        customerDao = new CustomerDaoOracleImpl ();
        contactDao = new ContactDaoOracleImpl();
    } else if (dbType.equals(SQLSERVER)) { /** 如果数据库类型为 SQL Server */
        customerDao = new CustomerDaoSqlServerImpl ();
        contactDao = new ContactDaoSqlServerImpl();
    }
}
```

这样修改后，调用不同数据库访问类就可以像以下代码：

```
DaoFactory daoFactory = new DaoFactory();
//设置数据库类型为 Oracle
daoFactory.setDbType(DaoFactory.ORACLE);
customerDao = daoFactory.getCustomerDao();
```

这样通过 setDbType 方法设置数据库的类型就可以获得不同数据访问对象。

25.4.5 数据库的初始化的脚本

本系统只以 MySQL 数据来实现，主要由两个表构成：CUSTOMER 表和 CONTACT 表。另外还需要初始化一些简单的数据。如果读者对 MySQL 数据不是很熟悉，请参阅其他的参考资料，这里只列举出 MySQL 的建表和初始化数据的 SQL 语句，不多作详细解释了。

MySQL.sql

```
-- 创建客户表
DROP TABLE IF EXISTS CUSTOMER;
CREATE TABLE CUSTOMER (
    CUSTOMER_ID      int(20) NOT NULL default '0',
    DISPLAY_NAME      varchar(255) NOT NULL default '',
    WEB_SITE          varchar(255) default NULL,
    CUSTOMER_CATEGORY varchar(64) default NULL,
    NUM_EMPLOYEE      varchar(255) default NULL,
    ACTIVE_STATUS     varchar(4) NOT NULL default 'Y'
) TYPE=MyISAM ;

-- 创建联系人表
DROP TABLE IF EXISTS CONTACT;
CREATE TABLE CONTACT(
    CONTACT_ID        int(20) NOT NULL default '0',
    CUSTOMER_ID        int(20) NOT NULL default '0',
```

```

DISPLAY_NAME      varchar(255) NOT NULL default "",
CONTACT_TYPE      varchar(16) NOT NULL default "",
SEX               varchar(64) default NULL,
BIRTH_DATE        varchar(64) default NULL,
JOB_TITLE         varchar(64) default NULL,
ACTIVE_STATUS     varchar(4) NOT NULL default 'Y'
) TYPE=MyISAM;

-- 初始化客户和联系人表数据
INSERT INTO CUSTOMER VALUES(1,'Nest Cafe','http://www.aaaa.com','Customer','51-100','Y');
INSERT INTO CUSTOMER VALUES(2,'Apache Foundation','http://www.bbbb.com','Customer',
'11-50','Y');
INSERT INTO CUSTOMER VALUES(3,'Sun System','http://www.cccc.com','Partener', '101-500',
'Y');

INSERT INTO CONTACT VALUES(1,1,'Janet','Friend','Female','8.5','CEO','Y');
INSERT INTO CONTACT VALUES(2,1,'Tina','Family','Male','6.6','CTO','Y');
INSERT INTO CONTACT VALUES(3,2,'Jone','Friend','Female','10.6','CIO','Y');

```

运行该 SQL 脚本后，数据库中的表如图 25.7 和图 25.8 所示。

CUSTOMER_ID	DISPLAY_NAME	WEB_SITE	CUSTOMER_CATEGORY	NUM_EMPLOYEE	ACTIVE_STATUS
1	Nest Cafe	http://www.aaaa.com	Customer	51-100	Y
2	Apache Foundation	http://www.bbbb.com	Customer	11-50	Y
3	Sun System	http://www.cccc.com	Partener	101-500	Y

图 25.7 CUSTOMER 表

CONTACT_ID	CUSTOMER_ID	DISPLAY_NAME	CONTACT_TYPE	SEX	BIRTH_DATE	JOB_TITLE	ACTIVE_STATUS
1	1	Janet	Friend	Female	8.5	CEO	Y
2	1	Tina	Family	Male	6.6	CTO	Y
3	2	Jone	Friend	Female	10.6	CIO	Y

图 25.8 CONTACT 表

25.4.6 表所对应的 POJO 类

数据库中的表一般要转化为 Java 所对应的 POJO 类，这样就将数据库中的记录转化为 Java 面向对象的类。与表对应，本系统中涉及两个 POJO 类：CustomerEO 和 ContactEO 类，分别对应表 CUSTOMER 和表 CONTACT。CustomerEO 和 ContactEO 的类图如图 25.9 和图 25.10 所示。

在此只列举出 CustomerEO 类的代码，而 ContactEO 类与之类似，只是属性不同而已。以下为 CustomerEO 类的代码。

CustomerEO.java

```

package superCRM.pojos;

public class CustomerEO {

```

```
/** 客户对象的属性 */
private int id;
private String displayName;
private String webSite;
private String category;
private String numberEmployee;
private String activeStatus = "Y";

/** 获得客户属性的 getter 和 setter 方法 */
public int getId() {
    return id;
}
public void setId(int id) {
    this.id = id;
}
public String getActiveStatus() {
    return activeStatus;
}
public void setActiveStatus(String activeStatus) {
    this.activeStatus = activeStatus;
}
public String getCategory() {
    return category;
}
public void setCategory(String category) {
    this.category = category;
}
public String getDisplayName() {
    return displayName;
}
public void setDisplayName(String displayName) {
    this.displayName = displayName;
}
public String getNumberEmployee() {
    return numberEmployee;
}
public void setNumberEmployee(String numberEmployee) {
    this.numberEmployee = numberEmployee;
}
public String getWebSite() {
    return webSite;
}
public void setWebSite(String webSite) {
    this.webSite = webSite;
}
}
```

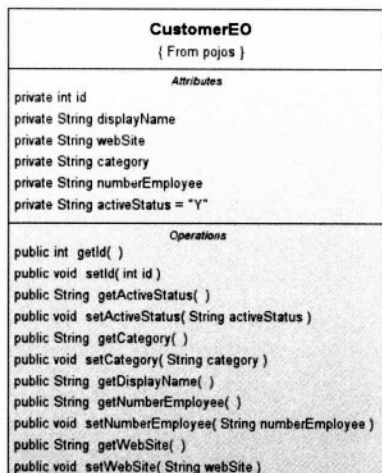


图 25.9 CustomerEO 类图

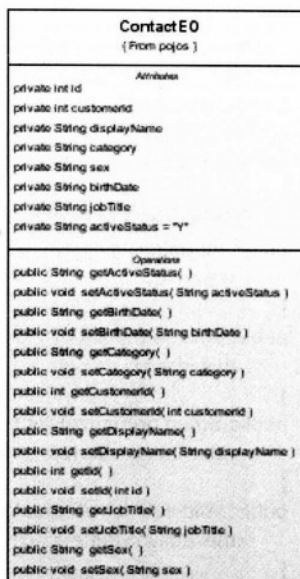


图 25.10 ContactEO 类图

25.5 登录模块

理解了业务层和数据层的设计后，下文就重点看一下 UI 界面部分如何实现。在运行系统之前，首先要进行用户验证，也就是该系统的登录模块。

25.5.1 系统的上下文对象保存登录状态

如何判断一个用户的登录状态呢？这里设计了一个系统的上下文类 `SuperContext`，该类仍然采用单例模式，只允许系统运行后创建一个该类的实例。并且通过布尔型的属性来保存用户登录的状态，如果成功登录，则设置该属性值为 `true`，否则设置为 `false`。

该系统上下文类的代码如下：

SuperContext.java

```
package superCRM.business;

public class SuperContext {

    /** 构造方法为 private */
    private SuperContext() {
    }
}
```

```
/** 单例模式的应用 */
private static SuperContext context = null;

public static SuperContext getInstance() {
    if (context == null)
        context = new SuperContext();
    return context;
}

/** 保存登录状态属性 */
private boolean bLogin = false;

/** 是否已登录 */
public boolean isLogin() {
    return bLogin;
}

/** 设置登录状态 */
public void setLogin(boolean login) {
    bLogin = login;
}
}
```

25.5.2 登录验证的实现

在 RCP 程序主窗口创建之前就需要对用户的身份进行验证。通过上文的学习，RCP 程序运行总是从入口对象开始的，所以要将验证用户的部分放在该类中，本系统 RCP 程序的入口类为 Application.java。添加用户判断逻辑后的代码如下：

Application.java

```
package superCRM.intro;

import org.eclipse.core.runtime.IPlatformRunnable;
import org.eclipse.jface.preference.IPreferenceStore;
import org.eclipse.jface.window.Window;
import org.eclipse.swt.widgets.Display;
import org.eclipse.ui.PlatformUI;

import superCRM.SuperCRMPlugin;
import superCRM.business.SuperContext;
import superCRM.dialog.LoginDialog;
import superCRM.preferences.PreferenceConstants;

/**
 * 该类为主程序的入口类
```



```

*/
public class Application implements IPlatformRunnable {

    /**
     * IPlatformRunnable 接口中的方法
     *
     * @see org.eclipse.core.runtime.IPlatformRunnable#run(java.lang.Object)
     */
    public Object run(Object args) throws Exception {
        Display display = PlatformUI.createDisplay();
        try {
            /**获得当前用户的上下对象*/
            SuperContext context = SuperContext.getInstance();
            /**调用 login 方法验证用户，如果用户名和密码不正确则退出程序*/
            if (!login(context))
                return IPlatformRunnable.EXIT_OK;
            /**验证成功后，创建主程序窗口*/
            int returnCode = PlatformUI.createAndRunWorkbench(display, new Application
WorkbenchAdvisor());
            if (returnCode == PlatformUI.RETURN_RESTART) {
                return IPlatformRunnable.EXIT_RESTART;
            }
            return IPlatformRunnable.EXIT_OK;
        } finally {
            display.dispose();
        }
    }

    /**
     * 判断用户是否已登录
     *
     * @param context 保存登录信息的上下文对象
     * @return true 表示已经登录
     */
    private boolean login(SuperContext context) {
        /**获得首选项中对登录的设置*/
        IPreferenceStore store = SuperCRMPugin.getDefault().getPreferenceStore();

        /**如果首选项设置为自动登录，则不需要登录*/
        if (store.getBoolean(PreferenceConstants.P_AUTO_LOGIN)){
            context.setLogin( true );
            /**否则，弹出登录对话框*/
        }else {
            LoginDialog loginDialog = new LoginDialog(null);
            /**如果输入了正确的用户名和密码，则设置登录成功*/
            if (loginDialog.open() == Window.OK)
                context.setLogin(true);
        }
        return context.isLogin();
    }
}

```

在验证登录时，也就是在 login 方法中，首先会判断用户首选项的设置，是否需要自动

登录。在使用一些聊天软件时，一般都会遇到这种情况。例如，在使用 MSN 时，系统会提示用户是否需要自动登录。当选择自动登录时，不需要输入密码和用户名就可以直接登录了。所以这里的逻辑也是这样的。

25.5.3 登录窗口的实现

当用户设置的首选项不是自动登录时，就需要弹出登录对话框，输入用户名和密码进行验证。如图 25.11 和图 25.12 所示为登录窗口的界面效果图。

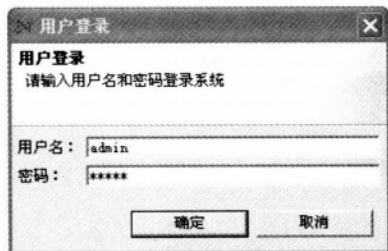


图 25.11 登录窗口图

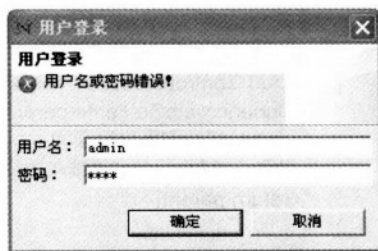


图 25.12 登录时错误提示的界面效果

该对话框类具体实现的代码如下：

LoginDialog.java

```
package superCRM.dialog;

import org.eclipse.jface.dialogs.IDialogConstants;
import org.eclipse.jface.dialogs.TitleAreaDialog;
import org.eclipse.jface.preference.IPreferenceStore;
import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Control;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.widgets.Text;
import superCRM.SuperCRMPlugin;
import superCRM.preferences.PreferenceConstants;
import superCRM.util.LayoutUtil;

/** 登录对话框 */
public class LoginDialog extends TitleAreaDialog {
    /** 用户名 */
    private Text userName;
    /** 密码 */
    private Text password;
```

```
public LoginDialog(Shell parentShell) {
    super(parentShell);
}
/** 设置登录对话框的属性 */
protected void configureShell(Shell newShell) {
    newShell.setText("用户登录");
    newShell.setSize(300, 200);

    newShell.setImage(SuperCRMPugin.getImageDescriptor("icons/logo.gif").createImage());
    LayoutUtil.centerShell(Display.getCurrent(), newShell);
}

/** 设置登录对话框的内容属性 */
protected Control createContents(Composite parent) {
    super.createContents(parent);
    this.setTitle("用户登录");
    this.setMessage("请输入用户名和密码登录系统");
    return parent;
}

/** 设置登录对话框内容区的属性 */
protected Control createDialogArea(Composite parent) {
    super.createDialogArea(parent);
    Composite composite = new Composite(parent, SWT.NONE);
    composite.setLayoutData(new GridData(GridData.FILL_BOTH));
    GridLayout layout = new GridLayout(2, false);
    composite.setLayout(layout);
    new Label(composite, SWT.NONE).setText("用户名: ");
    userName = new Text(composite, SWT.BORDER);
    userName.setLayoutData(new GridData(GridData.FILL_HORIZONTAL));
    new Label(composite, SWT.NONE).setText("密码: ");
    password = new Text(composite, SWT.BORDER);
    password.setEchoChar('*');
    password.setLayoutData(new GridData(GridData.FILL_HORIZONTAL));
    return parent;
}

/** 覆盖父类的方法，当单击按钮时调用 */
protected void buttonPressed(int buttonId) {
    /** 如果单击了“确定”按钮 */
    if (IDialogConstants.OK_ID == buttonId) {
        /** 用户名不为空 */
        if (userName.getText().equals("")) {
            this.setErrorMessage("用户名不为空");
            return;
        }
        /** 密码不为空 */
        if (password.getText().equals("")) {
            this.setErrorMessage("密码不为空!");
        }
    }
}
```

```

        return;
    }
    /**验证用户名和密码*/
    boolean bValid = checkValid();
    if (!bValid) {
        this.setErrorMessage("用户名或密码错误!");
        return;
    }
    okPressed();
} else if (IDialogConstants.CANCEL_ID == buttonId)
    cancelPressed();
}

/** 判断验证用户名和密码 */
private boolean checkValid() {
    boolean bValid = false;
    /**将用户输入用户名与首选项中设置的用户名和密码对比, 如果正确, 则验证成功*/
    IPreferenceStore store = SuperCRMPlugin.getDefault().getPreferenceStore();
    if (userName.getText().equals(store.getString(PreferenceConstants.P_USER_NAME))
    && password.getText().equals(store.getString(PreferenceConstants.P_USER_NAME)))
        bValid = true;
    return bValid;
}
}

```

验证用户名和密码的功能是在 `checkValid` 方法中实现的。本系统中用户名和密码是保存在首选项中, 也可以将其保存在数据中, 这里为了简单起见, 所以保存在首选项中。默认情况下, 登录的用户名和密码都为 `admin`。

在设置登录窗口的位置时, 使用了 `LayoutUtil` 中的 `centerShell` 方法来使登录窗口位于屏幕的中间位置。以下就为该工具类的代码:

LayoutUtil.java

```

package superCRM.util;

import org.eclipse.swt.graphics.Rectangle;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;

public class LayoutUtil {
    public static void centerShell(Display display, Shell shell) {
        Rectangle displayBounds = display.getPrimaryMonitor().getBounds();
        Rectangle shellBounds = shell.getBounds();
        int x = displayBounds.x + (displayBounds.width - shellBounds.width) >> 1;
        int y = displayBounds.y + (displayBounds.height - shellBounds.height) >> 1;
        shell.setLocation(x, y);
    }
}

```


25.6 主窗口界面

当验证通过后，将进入到窗口的主程序界面。下面就来看一下主窗口部分是如何实现的。

25.6.1 工作台的实现

工作台是通过创建 `ApplicationWorkbenchAdvisor` 类实现的，该类继承自 `WorkbenchAdvisor` 类，并且需要设置默认的透视图的 ID。

ApplicationWorkbenchAdvisor.java

```
package superCRM.intro;

import org.eclipse.ui.application.IWorkbenchConfigurer;
import org.eclipse.ui.application.IWorkbenchWindowConfigurer;
import org.eclipse.ui.application.WorkbenchAdvisor;
import org.eclipse.ui.application.WorkbenchWindowAdvisor;

/**工作区类*/
public class ApplicationWorkbenchAdvisor extends WorkbenchAdvisor {

    /**工作区默认的透视图 ID*/
    private static final String PERSPECTIVE_ID = "SuperCRM.perspective";

    /**创建工作区窗口*/
    public WorkbenchWindowAdvisor createWorkbenchWindowAdvisor(IWorkbenchWindow
    Configurer configurer) {
        return new ApplicationWorkbenchWindowAdvisor(configurer);
    }

    /**初始化工作区设置*/
    public void initialize(IWorkbenchConfigurer configurer) {
        super.initialize(configurer);
        configurer.setSaveAndRestore(true);
    }

    /**获得默认透视图的 ID*/
    public String getInitialWindowPerspectiveId() {
        return PERSPECTIVE_ID;
    }
}
```

该程序中默认的透视图 ID 为 `SuperCRM.perspective`，该透视图的配置请参阅 `plugin.xml` 文件中透视图配置部分。

25.6.2 系统托盘的实现

创建工作台后，将会创建工作台窗口对象 `ApplicationWorkbenchWindowAdvisor` 对象。同时也要在此处创建系统托盘。如图 25.13 所示为系统运行后，系统托盘的界面效果图。

以下是该工作台窗口类具体实现的代码：

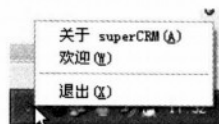


图 25.13 系统托盘界面效果

ApplicationWorkbenchWindowAdvisor.java

```
package superCRM.intro;

import org.eclipse.jface.action.MenuManager;
import org.eclipse.swt.SWT;
import org.eclipse.swt.graphics.Image;
import org.eclipse.swt.graphics.Point;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Event;
import org.eclipse.swt.widgets.Listener;
import org.eclipse.swt.widgets.Menu;
import org.eclipse.swt.widgets.Tray;
import org.eclipse.swt.widgets.TrayItem;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.application.ActionBarAdvisor;
import org.eclipse.ui.application.IActionBarConfigurer;
import org.eclipse.ui.application.IWorkbenchWindowConfigurer;
import org.eclipse.ui.application.WorkbenchWindowAdvisor;
import superCRM.SuperCRMPlugin;

/** 工作区窗口类 */

public class ApplicationWorkbenchWindowAdvisor extends WorkbenchWindowAdvisor {

    /** 系统托盘对象 */
    private TrayItem trayItem;

    /** 系统托盘的图标对象 */
    private Image trayImage;

    /** 程序的菜单栏 */
    private ApplicationActionBarAdvisor actionBarAdvisor;

    /** 构造方法 */
    public ApplicationWorkbenchWindowAdvisor(IWorkbenchWindowConfigurer configurer) {
        super(configurer);
    }

    /** 创建菜单栏对象 */
}
```

```

public ActionBarAdvisor createActionBarAdvisor(IActionBarConfigurer configurer) {
    actionBarAdvisor = new ApplicationActionBarAdvisor(configurer);
    return actionBarAdvisor;
}

/** 打开窗口前调用该方法，对窗口初始化设置 */
public void preWindowOpen() {
    /** 设置窗口初始化的各种属性 */
    IWorkbenchWindowConfigurer configurer = getWindowConfigurer();
    configurer.setInitialSize(new Point(700, 550));
    configurer.setShowCoolBar(true);
    configurer.setShowStatusLine(false);
    configurer.setTitle("SuperCRM");
    configurer.setShowPerspectiveBar(true);
    configurer.setShowStatusLine(true);

    final IWorkbenchWindow window = getWindowConfigurer().getWindow();
    /** 创建系统托盘 */
    trayItem = initTrayItem(window);
    /** 如果支持系统托盘，则创建托盘的菜单 */
    if (trayItem != null) {
        createPopupMenu(window);
    }
}

/**
 * 创建系统托盘菜单
 *
 * @param window
 * 工作台窗口对象
 */
private void createPopupMenu(final IWorkbenchWindow window) {
    trayItem.addListener(SWT.MenuDetect, new Listener() {
        public void handleEvent(Event event) {
            /** 通过 MenuManager 对象来创建菜单 */
            MenuManager trayMenu = new MenuManager();
            Menu menu = trayMenu.createContextMenu(window.getShell());

            /**
             * 调用 fillTrayItem 方法创建系统托盘对象，可以直接利用菜单栏中的操作，而
             不需要重新创建操作
             */
            actionBarAdvisor.fillTrayItem(trayMenu);
            menu.setVisible(true);
        }
    });
}

```

```

/**
 * 初始化系统托盘对象
 *
 * @param window
 * 工作台窗口对象
 * @return 该程序所对应的系统托盘对象
 */
private TrayItem initTrayItem(IWorkbenchWindow window) {
    final Tray tray = Display.getCurrent().getSystemTray();
    if (tray == null)
        return null;
    /** 创建系统托盘并设置系统托盘的各种属性 */
    TrayItem trayItem = new TrayItem(tray, SWT.NONE);
    trayImage = SuperCRMPlugin.getImageDescriptor("icons/logo.gif").createImage();
    trayItem.setImage(trayImage);
    trayItem.setToolTipText("SuperCRM");
    return trayItem;
}

/** 释放窗口，释放系统托盘 */
public void dispose() {
    if (trayImage != null) {
        trayImage.dispose();
        trayItem.dispose();
    }
}
}

```

创建系统托盘功能是通过 `preWindowOpen` 方法实现的。

25.6.3 菜单栏和工具栏的实现

创建工作台窗口对象时首先要创建窗口的菜单栏和工具栏的各种操作，如图 25.14、图 25.15 和图 25.16 所示为每个菜单项运行后的效果图。

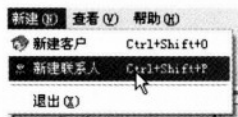


图 25.14 新建菜单项



图 25.15 查看菜单项



图 25.16 帮助菜单项

`ApplicationActionBarAdvisor` 类实现的具体代码如下：

ApplicationActionBarAdvisor.java

```
package superCRM.intro;
```

```

import org.eclipse.jface.action.ICoolBarManager;
import org.eclipse.jface.action.IMenuManager;
import org.eclipse.jface.action.IToolBarManager;
import org.eclipse.jface.action.MenuManager;
import org.eclipse.jface.action.Separator;
import org.eclipse.jface.action.ToolBarManager;
import org.eclipse.ui.IWorkbenchActionConstants;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.actions.ActionFactory;
import org.eclipse.ui.actions.ActionFactory.IWorkbenchAction;
import org.eclipse.ui.application.ActionBarAdvisor;
import org.eclipse.ui.application.IActionBarConfigurer;

import superCRM.action.ActionManager;
import superCRM.views.ContactSummaryView;
import superCRM.views.CustomerSummaryView;
import superCRM.views.NavView;
import superCRM.views.QuickNewContactView;
import superCRM.views.QuickNewCustomerView;
import superCRM.views.SearchView;

/** 窗口的菜单栏类 */
public class ApplicationActionBarAdvisor extends ActionBarAdvisor {

    /** 新建菜单的菜单项 */
    private IWorkbenchAction newCustomerAction;
    private IWorkbenchAction newContactAction;
    private IWorkbenchAction exitAction;
    /** 查看菜单的菜单项 */
    private IWorkbenchAction viewNavAction;
    private IWorkbenchAction viewSearchAction;
    private IWorkbenchAction viewFastNewCustomerAction;
    private IWorkbenchAction viewFastNewCintactAction;
    private IWorkbenchAction viewCustomerAction;
    private IWorkbenchAction viewContactAction;
    /** 帮助菜单的菜单项 */
    private IWorkbenchAction introAction;
    private IWorkbenchAction helpAction;
    private IWorkbenchAction aboutAction;
    private IWorkbenchAction preferenceAction;
    /** 构造方法 */
    public ApplicationActionBarAdvisor(IActionBarConfigurer configurer) {
        super(configurer);
    }

    protected void makeActions(IWorkbenchWindow window) {
        /** 新建客户操作 */
        newCustomerAction = ActionManager.createNewCustomerAction(window);
        register(newCustomerAction);
    }

```



```
/** 新建联系人操作 */
newContactAction = ActionManager.createNewContactAction(window);
register(newContactAction);

/**退出操作*/
exitAction = ActionFactory.QUIT.create(window);
register(exitAction);

/**查看导航视图操作*/
viewNavAction = ActionManager.createShowViewAction(window, NavView.ID);
register(viewNavAction);

/**查看搜索视图操作*/
viewSearchAction = ActionManager.createShowViewAction(window, SearchView.ID);
register(viewSearchAction);

/**查看快速新建客户视图操作*/
viewFastNewCustomerAction = ActionManager.createShowViewAction(window, QuickNew
CustomerView.ID);
register(viewFastNewCustomerAction);

/**查看快速新建联系人视图操作*/
viewFastNewCintactAction = ActionManager.createShowViewAction(window, QuickNew
ContactView.ID);
register(viewFastNewCintactAction);

/**查看客户列表视图操作*/
viewCustomerAction = ActionManager.createShowViewAction(window, Customer
SummaryView.ID);
register(viewCustomerAction);

/**查看联系人列表视图操作*/
viewContactAction = ActionManager.createShowViewAction(window, ContactSummary
View.ID);
register(viewContactAction);

/**打开欢迎视图操作*/
introAction = ActionFactory.INTRO.create(window);
register(introAction);

/**打开帮助窗口操作*/
helpAction = ActionFactory.HELP_CONTENTS.create(window);
register(helpAction);

/**打开关于窗口操作*/
aboutAction = ActionFactory.ABOUT.create(window);
register(aboutAction);
```



```
/**打开首选项操作*/
preferenceAction = ActionFactory.PREFERENCES.create(window);
register(preferenceAction);
}

/**
 * 创建菜单栏，并添加菜单项
 */
protected void fillMenuBar(IMenuManager menuBar) {

    MenuManager systemMenu = new MenuManager("新建(&N)");
    systemMenu.add(new CustomerAction());
    systemMenu.add(new ContactAction());
    systemMenu.add(new Separator());
    systemMenu.add(exitAction);
    menuBar.add(systemMenu);

    MenuManager viewMenu = new MenuManager("查看(&V)");
    viewMenu.add(viewNavAction);
    viewMenu.add(viewSearchAction);
    viewMenu.add(new Separator());
    viewMenu.add(viewFastNewCustomerAction);
    viewMenu.add(viewFastNewContactAction);
    viewMenu.add(new Separator());
    viewMenu.add(viewCustomerAction);
    viewMenu.add(viewContactAction);
    menuBar.add(viewMenu);

    MenuManager helpMenu = new MenuManager("帮助(&H)", IWorkbenchActionConstants.
M_HELP);
    helpMenu.add(introAction);
    helpMenu.add(helpAction);
    helpMenu.add(aboutAction);
    helpMenu.add(preferenceAction);
    menuBar.add(helpMenu);
}

/**
 * 创建工具栏，并添加工具按钮
 */
protected void fillCoolBar(ICoolBarManager coolBar) {
    IToolBarManager barManager = new ToolBarManager(coolBar.getStyle());
    barManager.add(new CustomerAction());
    barManager.add(new ContactAction());
    barManager.add(new Separator());
    barManager.add(viewNavAction);
    barManager.add(viewSearchAction);

    barManager.add(viewFastNewCustomerAction);
    barManager.add(viewFastNewContactAction);
}
```

```

        barManager.add(viewCustomerAction);
        barManager.add(viewContactAction);
        coolBar.add(barManager);
    }
    /**
     * 添加系统托盘所使用的操作项
     *
     * @param trayMenu
     */
    public void fillTrayItem(MenuManager trayMenu) {
        trayMenu.add(aboutAction);
        trayMenu.add(introAction);
        trayMenu.add(new Separator());
        trayMenu.add(exitAction);
    }
}

```

读者应该对该程序的代码并不陌生，所有的 Action 操作都可以添加到菜单项、工具按钮和系统托盘菜单项中。

25.6.4 操作管理类（ActionManager）

仿照 ActionFactory 类中创建操作的方法，在该系统中将所涉及的 Action 类也使用 ActionManager 类来进行创建。这样设计的好处是，能够方便地获得各个操作的实例。

ActionManager 类的具体代码如下：

ActionManager.java

```

package superCRM.action;

import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.actions.ActionFactory.IWorkbenchAction;
import org.eclipse.ui.views.IViewDescriptor;

public class ActionManager {

    /** 新建客户操作 */
    public static IWorkbenchAction createNewCustomerAction(IWorkbenchWindow window) {
        if (window == null) {
            throw new IllegalArgumentException();
        }
        IWorkbenchAction action = new NewCustomerAction(window);
        return action;
    }

    /** 新建联系人操作 */
    public static IWorkbenchAction createNewContactAction(IWorkbenchWindow window) {

```

```

        if (window == null) {
            throw new IllegalArgumentException();
        }
        IWorkbenchAction action = new NewContactAction(window);
        return action;
    }
    /**打开视图操作 */
    public static IWorkbenchAction createShowViewAction(IWorkbenchWindow window, String
viewId) {
        if (window == null) {
            throw new IllegalArgumentException();
        }
        /**获得 plugin.xml 文件中配置的视图信息*/
        IViewDescriptor desc = window.getWorkbench().getViewRegistry().find(viewId);
        IWorkbenchAction action = new ShowViewAction(window, desc);
        return action;
    }
}

```

该类主要有 3 个创建 Action 的静态方法，分别用于创建不同操作。

25.6.5 新建客户操作（NewCustomerAction）

在 ActionManager 类中，通过 createNewCustomerAction 方法可以直接创建新建客户对象 NewCustomerAction。当执行该操作时，将会弹出新建客户对话框。

NewCustomerAction 类的具体代码如下：

NewCustomerAction.java

```

package superCRM.action;

import org.eclipse.jface.action.Action;
import org.eclipse.jface.wizard.WizardDialog;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.actions.ActionFactory.IWorkbenchAction;

import superCRM.SuperCRMPlugin;
import superCRM.dialog.NewCustomerWizard;

public class NewCustomerAction extends Action implements IWorkbenchAction {
    /** 操作所对应的 ID */
    public static final String ID = "superCRM.action.NewCustomerAction";

    private IWorkbenchWindow workbenchWindow;

    public NewCustomerAction(IWorkbenchWindow window) {
        if (window == null) {
            throw new IllegalArgumentException();
        }
    }
}

```

```
    }
    this.workbenchWindow = window;
    /** 设置 ID */
    setId(ID);
    setActionDefinitionId(ID);
    setText("新建客户");
    setImageDescriptor(SuperCRMPlugin.getImageDescriptor("icons/customer.gif"));
}

public void run() {
    if (workbenchWindow == null) {
        return;
    }
    /** 打开新建客户对话框 */
    WizardDialog dlg = new WizardDialog(workbenchWindow.getShell(), new NewCustomer
Wizard(workbenchWindow));
    dlg.open();
}

public void dispose() {
    workbenchWindow = null;
}
}
```

在构造方法中，setId 方法设置该操作所对应的 ID，是因为在 plugin.xml 文件中设置了该 ID 所对应的快捷键。读者可以参阅 plugin.xml 文件中的配置的"org.eclipse.ui.commands"扩展部分。这样就可以通过配置的快捷键来调用该操作了。

另外，新建客户操作 NewContactAction 类，与新建客户操作类似。只是执行时调用新建联系人 NewContactWizard 向导对话框，所以这里就不详细说明该类的代码部分了。

25.6.6 打开视图操作（ShowViewAction）

因为本系统中涉及多个视图，如果为每一个打开视图的操作都编写一个 Action 类，这样会有大量重复的代码。所以这里将打开视图的操作编写成一个类，这样只要传入一个视图的 ID，就可以获得打开视图的操作对象。

打开视图 ShowViewAction 类的具体代码如下：

ShowViewAction.java

```
package superCRM.action;

import org.eclipse.jface.action.Action;
import org.eclipse.jface.dialogs.ErrorDialog;
import org.eclipse.ui.IWorkbenchPage;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.PartInitException;
import org.eclipse.ui.actions.ActionFactory.IWorkbenchAction;
```



```
import org.eclipse.ui.views.IViewDescriptor;

public class ShowViewAction extends Action implements IWorkbenchAction {

    private IWorkbenchWindow workbenchWindow;

    private IViewDescriptor desc;

    public ShowViewAction(IWorkbenchWindow window, IViewDescriptor desc) {
        super("");
        /** 获得视图的名称 */
        String label = desc.getLabel();
        /** 设置操作的名称 */
        setText(label);
        /** 设置操作的图标为视图的图标 */
        setImageDescriptor(desc.getImageDescriptor());
        /** 设置操作的提示文本 */
        setToolTipText(label);
        setId("ShowView" + desc.getId());
        this.workbenchWindow = window;
        this.desc = desc;
    }

    public void run() {
        /** 获得当前工作区及获得工作页面 */
        IWorkbenchPage page = workbenchWindow.getActivePage();
        /** 如果页面不为 null */
        if (page != null) {
            try {
                /** 显示视图 */
                page.showView(desc.getId());
            } catch (PartInitException e) {
                ErrorDialog.openError(workbenchWindow.getShell(), "打开视图错误!", e.getMessage(), e.getStatus());
            }
        }
    }

    public void dispose() {
        workbenchWindow = null;
    }
}
```

25.7 各种视图和编辑器的实现

视图和编辑器是构成该客户关系管理系统的主要的界面部分，是整个系统最重要的部分。视图和编辑器创建的步骤在上文中已经详细讲述过，所以这部分内容主要是讲述视图

和编辑器的代码。

25.7.1 快速新建客户视图

用户可以在快速新建客户视图中快速新建一个客户记录。添加新的客户后，将会自动刷新客户列表视图，将新建的客户记录显示在列表中。如图 25.17 所示为快速新建客户视图的界面效果图。



图 25.17 快速新建客户视图

快速新建客户视图 QuickNewCustomerView 的代码如下：

QuickNewCustomerView.java

```
package superCRM.views;

import org.eclipse.swt.SWT;
import org.eclipse.swt.events.SelectionAdapter;
import org.eclipse.swt.events.SelectionEvent;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Combo;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.Text;
import org.eclipse.ui.IViewPart;
import org.eclipse.ui.PartInitException;
import org.eclipse.ui.part.ViewPart;

import superCRM.model.ICustomerService;
import superCRM.model.SuperFactory;
import superCRM.pojos.CustomerEO;

public class QuickNewCustomerView extends ViewPart {
    public static final String ID = "superCRM.views.QuickNewCustomerView";

    /** 客户名称 */
    private Text displayName;

    /** 网址 */
    private Text website;

    /** 客户分类 */
    private Combo category;

    /** 公司人数 */
    private Combo number;
```

```
public QuickNewCustomerView() {
    super();
}

/** 创建视图的界面 */
public void createPartControl(Composite parent) {
    /** 设置面板布局 */
    Composite composite = new Composite(parent, SWT.NONE);
    composite.setLayout(new GridLayout(2, false));
    GridData data = new GridData(GridData.FILL_HORIZONTAL);
    /** 客户名称 */
    new Label(composite, SWT.NONE).setText("客户名称: ");
    displayName = new Text(composite, SWT.BORDER);
    displayName.setLayoutData(data);
    /** 网址 */
    new Label(composite, SWT.NONE).setText("网址: ");
    website = new Text(composite, SWT.BORDER);
    data = new GridData(GridData.FILL_HORIZONTAL);
    website.setLayoutData(data);
    /** 客户分类 */
    new Label(composite, SWT.NONE).setText("类别: ");
    category = new Combo(composite, SWT.BORDER);
    category.setItems(new String[] { "Customer", "Partener", "Competitor" });
    category.select(0);
    data = new GridData(GridData.FILL_HORIZONTAL);
    category.setLayoutData(data);
    /** 公司人数 */
    new Label(composite, SWT.NONE).setText("公司人数: ");
    number = new Combo(composite, SWT.BORDER);
    number.setItems(new String[] { "1-10", "11-50", "51-100", "101-500", "500-" });
    number.select(0);
    data = new GridData(GridData.FILL_HORIZONTAL);
    number.setLayoutData(data);
    /** 添加按钮 */
    Button btAdd = new Button(composite, SWT.PUSH);
    btAdd.setText("添加");
    /** 注册添加按钮监听器 */
    btAdd.addSelectionListener(new SelectionAdapter() {
        /** 当单击“添加”按钮时 */
        public void widgetSelected(SelectionEvent e) {
            /** 创建客户对象 */
            CustomerEO customer = new CustomerEO();
            /** 分别取出用户输入值，并赋值给客户对象 */
            customer.setDisplayName(displayName.getText());
            customer.setWebSite(website.getText());
            customer.setCategory(category.getText());
            customer.setNumberEmployee(number.getText());
            /** 调用业务层保存客户数据 */
            ICustomerService customerService = SuperFactory.getSuperApplication().get
```

```

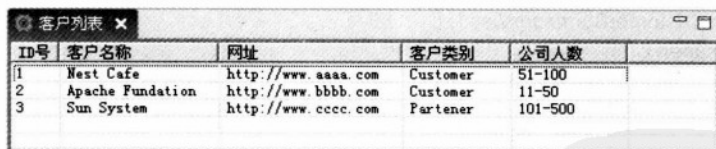
CustomerService();
    customerService.addCustomer(customer);
    /** 查找是否客户列表视图已经打开 */
    IViewPart view = getViewSite().getPage().findView (CustomerSummaryView. ID);
    /** 若没打开, 则首先打开客户列表视图 */
    if (view == null) {
        try {
            view=getViewSite().getPage().showView(CustomerSummaryView. ID);
        } catch (PartInitException ee) {
            ee.printStackTrace();
        }
    }
    /** 更新客户列表数据 */
    CustomerSummaryView customerSummaryView = (CustomerSummaryView)
view;
    customerSummaryView.refreshData();
}
});
}
public void setFocus() {
    displayName.setFocus();
}
}
}

```

该程序中, 视图的界面部分是通过 SWT 中的各种控件创建的。程序应该注意的地方是如何在单击“添加”按钮后, 调用业务逻辑将用户输入的数据保存在数据库中。

25.7.2 客户列表视图

客户列表视图显示了客户的列表信息, 可以显示查询的结果。如图 25.18 所示为客户列表视图的界面效果图。



ID号	客户名称	网址	客户类别	公司人数
1	Nest Cafe	http://www.aaaa.com	Customer	51-100
2	Apache Foundation	http://www.bbbb.com	Customer	11-50
3	Sun System	http://www.cccc.com	Partener	101-500

图 25.18 客户列表视图界面效果图

客户列表视图类 CustomerSummaryView 实现的具体代码如下:

CustomerSummaryView.java

```

package superCRM.views;

import java.util.List;

import org.eclipse.jface.viewers.DoubleClickEvent;

```

```

import org.eclipse.jface.viewers.IDoubleClickListener;
import org.eclipse.jface.viewers.StructuredSelection;
import org.eclipse.jface.viewers.TableViewer;
import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.TableColumn;
import org.eclipse.ui.IWorkbenchPage;
import org.eclipse.ui.PartInitException;
import org.eclipse.ui.part.ViewPart;

import superCRM.editor.CustomerDetailEditor;
import superCRM.editor.CustomerDetailInput;
import superCRM.model.ICustomerService;
import superCRM.model.SuperFactory;
import superCRM.pojos.CustomerEO;
import superCRM.table.TableContentProvider;
import superCRM.table.TableLabelProvider;

public class CustomerSummaryView extends ViewPart {
    /** 该视图的 ID */
    public static final String ID = "superCRM.views.CustomerSummaryView";

    /** 表头数据 */
    public static final String[] COLUMN_NAME = {"ID 号", "客户名称", "网址", "客户类别", "公司人数"};

    /** 表中的数据 */
    public List data;
    /** 表对象 */
    private TableViewer viewer;
    /** 客户服务对象 */
    private ICustomerService customerService;
    /** 客户详细编辑器输入对象 */
    private CustomerDetailInput customerDetailInput;
    public CustomerSummaryView() {
        super();
        /** 获得客户服务对象 */
        customerService = SuperFactory.getSuperApplication().getCustomerService();
        /** 查询所有的客户数据 */
        data = customerService.getAllCustomers();
    }

    /** 创建视图的界面 */
    public void createPartControl(Composite parent) {
        /** 初始化表格 */
        viewer = new TableViewer(parent, SWT.FULL_SELECTION);
        for (int i = 0; i < COLUMN_NAME.length; i++) {
            new TableColumn(viewer.getTable(), SWT.LEFT).setText(COLUMN_NAME[i]);
            viewer.getTable().getColumn(i).pack();
        }
    }
}

```



```

viewer.getTable().setHeaderVisible(true);
viewer.getTable().setLinesVisible(true);
/** 设置表格的内容器 */
viewer.setContentProvider(new TableContentProvider());
/** 设置表格的标签器 */
viewer.setLabelProvider(new TableLabelProvider());
/** 设置表格的数据 */
viewer.setInput(data);
/** 当双击表格时 */
viewer.addClickListener(new IDoubleClickListener() {

    public void doubleClick(DoubleClickEvent event) {
        /** 打开客户详细编辑器 */
        StructuredSelection select = (StructuredSelection) event.getSelection();
        /** 获得当前选中的客户 */
        CustomerEO c = (CustomerEO) select.getFirstElement();
        /** 创建客户详细编辑器输入对象 */
        customerDetailInput = new CustomerDetailInput(c);
        /** 打开客户详细编辑器 */
        IWorkbenchPage page = getViewSite().getWorkbenchWindow().GetActive
Page();

        try {
            page.openEditor(customerDetailInput, CustomerDetailEditor.ID);
        } catch (PartInitException e) {
            e.printStackTrace();
        }
    }
});
/** 将该表格注册为选择服务提供者 */
this.getSite().setSelectionProvider(viewer);
}

public void setFocus() {
    viewer.getTable().setFocus();
}

public List getData() {
    return data;
}

/** 设置表格数据 */
public void setData(List data) {
    this.data = data;
    viewer.setInput(this.data);
    viewer.refresh();
}

/** 刷新表格数据 */
public void refreshData() {
    data = customerService.getAllCustomers();
    viewer.setInput(data);
    viewer.refresh();
}

```



```
    }  
}
```

在创建显示的客户信息的表格 TableViewer 对象的内容器和标签器是通过 TableContent Provider 对象和 TableLabelProvider 对象来创建的。

TableContentProvider 和 TableLabelProvider 不仅用来设置客户表格的内容器和标签器, 也负责设置联系人表格的内容器和标签器。

TableContentProvider 类的代码如下:

TableContentProvider.java

```
package superCRM.table;  
  
import java.util.List;  
import org.eclipse.jface.viewers.IStructuredContentProvider;  
import org.eclipse.jface.viewers.Viewer;  
  
public class TableContentProvider implements IStructuredContentProvider {  
    public Object[] getElements(Object inputElement) {  
        /** 如果输入的是 List 对象 */  
        if (inputElement instanceof List) {  
            List list = (List) inputElement;  
            return list.toArray();  
        }  
        return null;  
    }  
    public void dispose() {}  
    public void inputChanged(Viewer viewer, Object oldInput, Object newInput){}  
}
```

TableLabelProvider 类的代码如下:

TableLabelProvider.java

```
package superCRM.table;  
  
import org.eclipse.jface.viewers.ILabelProviderListener;  
import org.eclipse.jface.viewers.ITableLabelProvider;  
import org.eclipse.swt.graphics.Image;  
  
import superCRM.model.ICustomerService;  
import superCRM.model.SuperFactory;  
import superCRM.pojos.ContactEO;  
import superCRM.pojos.CustomerEO;  
  
public class TableLabelProvider implements ITableLabelProvider {  
  
    public Image getColumnImage(Object element, int columnIndex) {  
        return null;  
    }  
}
```

```

/** 设置表格显示的数据 */
public String getColumnText(Object element, int columnIndex) {
    /** 如果输入的是客户对象, 返回相应的数据值 */
    if (element instanceof CustomerEO) {
        CustomerEO c = (CustomerEO) element;
        if (columnIndex == 0)
            return c.getId() + " ";
        else if (columnIndex == 1)
            return c.getDisplayName();
        else if (columnIndex == 2)
            return c.getWebSite();
        else if (columnIndex == 3)
            return c.getCategory();
        else if (columnIndex == 4)
            return c.getNumberEmployee();
        /** 如果输入的是联系人对象, 返回相应的数据值 */
    } else if (element instanceof ContactEO) {
        ContactEO c = (ContactEO) element;

        if (columnIndex == 0)
            return c.getId() + " ";
        else if (columnIndex == 1)
            return c.getDisplayName();
        else if (columnIndex == 2) {
            /** 如果是客户的 ID, 要查询该客户的名称, 并将名称显示 */
            ICustomerService customerService = SuperFactory.getSuperApplication().getCustomerService();
            CustomerEO customer = customerService.getCustomer(c.getCustomerId());
            if (customer != null)
                return customer.getDisplayName();
            return " ";
        } else if (columnIndex == 3)
            return c.getCategory();
        else if (columnIndex == 4)
            return c.getSex();
        else if (columnIndex == 5)
            return c.getBirthDate();
        else if (columnIndex == 6)
            return c.getJobTitle();
        }
    }
    return null;
}

public void addListener(ILabelProviderListener listener) {
}

public void dispose() {
}

public boolean isLabelProperty(Object element, String property) {
    return false;
}

```

```

    }
    public void removeListener(ILabelProviderListener listener) {
    }
}

```

25.7.3 客户详细编辑器

当双击客户列表时，会打开相应的客户详细编辑器。如图 25.19 所示为打开的两个客户的编辑器效果图。其中带星号的为已修改后的客户信息。

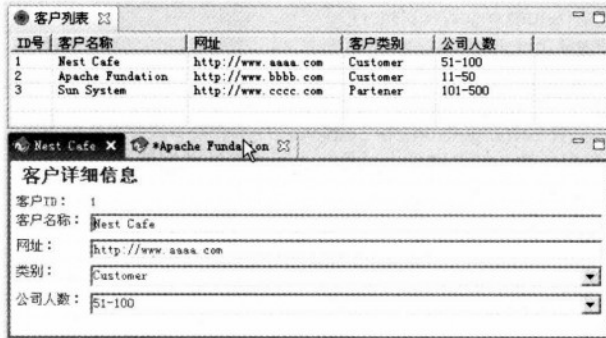


图 25.19 客户详细编辑器界面效果图

在客户详细编辑器中，可以查看客户的信息，也可以修改客户的信息。另外，编辑器中的控件是通过 Eclipse 表单来实现的，这样界面看起来更加友好。

客户详细编辑器 CustomerDetailEditor 的具体代码如下：

CustomerDetailEditor.java

```

package superCRM.editor;

import org.eclipse.core.runtime.IProgressMonitor;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.ModifyEvent;
import org.eclipse.swt.events.ModifyListener;
import org.eclipse.swt.widgets.Combo;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Text;
import org.eclipse.ui.IEditorInput;
import org.eclipse.ui.IEditorPart;
import org.eclipse.ui.IEditorSite;
import org.eclipse.ui.IViewPart;
import org.eclipse.ui.PartInitException;
import org.eclipse.ui.forms.widgets.FormToolkit;
import org.eclipse.ui.forms.widgets.ScrolledForm;
import org.eclipse.ui.forms.widgets.TableWrapData;
import org.eclipse.ui.forms.widgets.TableWrapLayout;

```

```
import org.eclipse.ui.part.EditorPart;

import superCRM.model.ICustomerService;
import superCRM.model.SuperFactory;
import superCRM.pojos.CustomerEO;
import superCRM.views.CustomerSummaryView;

public class CustomerDetailEditor extends EditorPart {
    /** 编辑器的 ID */
    public static final String ID = "superCRM.editor.CustomerDetailEditor";

    /** 编辑器是否修改过的标记 */
    private boolean bDirty = false;

    /** 编辑器保存的客户对象 */
    private CustomerEO customer;

    /** 界面控件 */
    private Text displayName;
    private Text website;
    private Combo category;
    private Combo number;
    /** 表单工具对象 */
    private FormToolkit toolkit;

    public CustomerDetailEditor() {
        super();
    }

    public void init(IEditorSite site, IEditorInput input) throws PartInitException {
        this.setSite(site);
        this.setInput(input);
        this.setPartName(input.getName());
        /** 将输入的编辑器对象的客户对象赋值给客户对象 */
        this.customer = ((CustomerDetailInput) input).getCustomer();
    }

    public void createPartControl(Composite parent) {
        /** 设置表单的基本属性 */
        toolkit = new FormToolkit(parent.getDisplay());
        ScrolledForm form = toolkit.createScrolledForm(parent);
        TableWrapLayout tableLayout = new TableWrapLayout();
        tableLayout.numColumns = 2;
        form.getBody().setLayout(tableLayout);
        form.setText("客户详细信息");
        toolkit.getColors().setForeground(form.getForeground());
        /** 创建界面的控件 */
        toolkit.createLabel(form.getBody(), "客户 ID: ");
        toolkit.createLabel(form.getBody(), customer.getId() + " ");
    }
}
```



```

        toolkit.createLabel(form.getBody(), "客户名称: ");
        TableWrapData data = new TableWrapData(TableWrapData.FILL_GRAB);
        /** 客户名称 */
        displayName = toolkit.createText(form.getBody(), customer.getDisplayName(), SWT.
BORDER);
        displayName.setLayoutData(data);
        ModifyListener listener = new ModifyListener() {
            public void modifyText(ModifyEvent e) {
                if (!isDirty()) {
                    bDirty = true;
                    firePropertyChange(IEditorPart.PROP_DIRTY);
                }
            }
        };
        displayName.addModifyListener(listener);

        /** 网址 */
        toolkit.createLabel(form.getBody(), "网址: ");
        data = new TableWrapData(TableWrapData.FILL_GRAB);
        website = toolkit.createText(form.getBody(), customer.getWebSite(), SWT.BORDER);
        website.setLayoutData(data);
        website.addModifyListener(listener);

        /** 类别 */
        toolkit.createLabel(form.getBody(), "类别: ");
        data = new TableWrapData(TableWrapData.FILL_GRAB);
        category = new Combo(form.getBody(), SWT.BORDER);
        toolkit.adapt(category, false, false);
        category.setItems(new String[] { "Customer", "Partener", "Competitor" });
        category.setLayoutData(data);
        category.setText(customer.getCategory());
        category.addModifyListener(listener);

        /** 公司人数 */
        toolkit.createLabel(form.getBody(), "公司人数: ");
        data = new TableWrapData(TableWrapData.FILL_GRAB);
        number = new Combo(form.getBody(), SWT.BORDER);
        toolkit.adapt(number, false, false);
        number.setItems(new String[] { "1-10", "11-50", "51-100", "101-500", "500-" });
        number.setLayoutData(data);
        number.setText(customer.getNumberEmployee());
        number.addModifyListener(listener);
    }

    /** 保存编辑器 */
    public void doSave(IProgressMonitor monitor) {
        try {
            monitor.beginTask("保存客户...", 100);
            /** 获得修改后的客户信息 */

```



```

        customer.setDisplayName(displayName.getText());
        customer.setWebSite(website.getText());
        customer.setCategory(category.getText());
        customer.setNumberEmployee(number.getText());
        /** 调用业务层保存更新客户信息 */
        ICustomerService customerService = SuperFactory.getSuperApplication().get
CustomerService();
        customerService.updateCustomer(customer);
        /** 更新客户列表视图数据 */
        refreshView();
        monitor.done();
        if (monitor.isCanceled())
            throw new InterruptedException("取消保存");
    } catch (InterruptedException e) {
        ;
    }
}

public void doSaveAs() {
}

public boolean isDirty() {
    return bDirty;
}

public boolean isSaveAsAllowed() {
    return false;
}

public void setFocus() {
    displayName.setFocus();
}

/** 更新客户列表视图数据 */
public void refreshView() {
    IViewPart view = getEditorSite().getPage().findView(CustomerSummaryView.ID);
    if (view == null)
        return;
    CustomerSummaryView customerSummaryView = (CustomerSummaryView) view;
    customerSummaryView.refreshData();
}
}

```

另外，在打开客户详细编辑器时，需要创建输入的编辑器 CustomerDetailInput 对象。该类具体的代码如下：

CustomerDetailInput.java

```
package superCRM.editor;
```

```
import org.eclipse.jface.resource.ImageDescriptor;
import org.eclipse.ui.IEditorInput;
import org.eclipse.ui.IPersistableElement;

import superCRM.pojos.CustomerEO;
import superCRM.SuperCRMPlugin;

public class CustomerDetailInput implements IEditorInput {

    /** 用户保存输入的客户信息 */
    private CustomerEO customer;

    public CustomerDetailInput(CustomerEO c) {
        this.customer = c;
    }

    /** 以下为接口中的方法 */
    public boolean exists() {
        return true;
    }

    public ImageDescriptor getImageDescriptor() {
        return SuperCRMPlugin.getImageDescriptor("icons/customer.gif");
    }

    public String getName() {
        return customer.getDisplayName();
    }

    public IPersistableElement getPersistable() {
        return null;
    }

    public String getToolTipText() {
        return customer.getDisplayName();
    }

    public Object getAdapter(Class adapter) {
        return null;
    }

    /** 获得客户对象 */
    public CustomerEO getCustomer() {
        return customer;
    }

    /** 设置客户对象 */
    public void setCustomer(CustomerEO customer) {
```

```

        this.customer = customer;
    }
}

```

25.7.4 联系人列表视图

当单击客户列表时,若此时联系视图已经打开,则随着光标选中的客户不同,在联系人列表视图中会显示该客户所属的联系人,如图 25.20 所示。

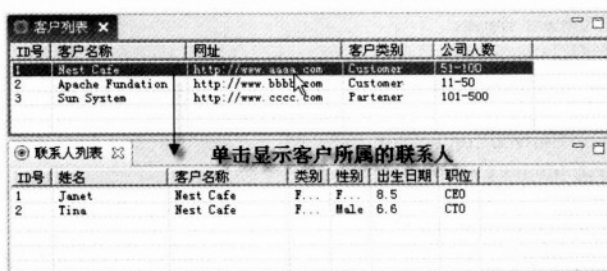


图 25.20 联系人列表视图界面效果图

联系人列表视图 ContactSummaryView 类的具体代码如下:

ContactSummaryView.java

```

package superCRM.views;

import java.util.List;

import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.viewers.IStructuredSelection;
import org.eclipse.jface.viewers.TableViewer;
import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.TableColumn;
import org.eclipse.ui.ISelectionListener;
import org.eclipse.ui.IWorkbenchPart;
import org.eclipse.ui.part.ViewPart;

import superCRM.model.IContactService;
import superCRM.model.ICustomerService;
import superCRM.model.SuperFactory;
import superCRM.pojos.CustomerEO;
import superCRM.table.TableContentProvider;
import superCRM.table.TableLabelProvider;

public class ContactSummaryView extends ViewPart implements ISelectionListener {
    /** 该视图的 ID */

```

```

public static final String ID = "superCRM.views.ContactSummaryView";
/** 表头数据 */
public static final String[] COLUMN_NAME = {"ID 号", "姓名", "客户名称", "类别", "性别", "出生日期", "职位"};
/** 当前选中的对象 */
private IStructuredSelection currentSelection;
/** 联系人数据所属的客户 */
private CustomerEO customer;
/** 联系人数据对象 */
public List data;
/** 联系人表对象 */
private TableViewer viewer;
/** 客户服务 */
private ICustomerService customerService;
/** 联系人服务 */
private IContactService contactService;
public ContactSummaryView() {
    super();
    /** 创建客户和联系人服务对象 */
    customerService = SuperFactory.getSuperApplication().getCustomerService();
    contactService = SuperFactory.getSuperApplication().getContactService();
}

public void createPartControl(Composite parent) {
    /** 注册选择服务对象 */
    this.getSite().getWorkbenchWindow().getSelectionService().addSelectionListener(this);
    /** 初始化联系人表格 */
    viewer = new TableViewer(parent, SWT.FULL_SELECTION);
    for (int i = 0; i < COLUMN_NAME.length; i++) {
        new TableColumn(viewer.getTable(), SWT.LEFT).setText(COLUMN_NAME[i]);
        viewer.getTable().getColumn(i).pack();
    }
    viewer.getTable().setHeaderVisible(true);
    viewer.getTable().setLinesVisible(true);
    viewer.setContentProvider(new TableContentProvider());
    viewer.setLabelProvider(new TableLabelProvider());
    /** 默认情况下设置显示所有联系人 */
    data = contactService.getAllContacts();
    viewer.setInput(data);
}

public void setFocus() {
    viewer.getTable().setFocus();
}

/** 当单击客户记录时 */
public void selectionChanged(IWorkbenchPart part, ISelection selection) {
    if (selection instanceof IStructuredSelection) {

```



```

        /** 获得当前选中的客户对象 */
        currentSelection = (IStructuredSelection) selection;
        if (currentSelection.getFirstElement() instanceof CustomerEO) {
            customer = (CustomerEO) currentSelection.getFirstElement();
            /** 重新设置联系人列表中的数据 */
            this.setData(customerService.getContacts(customer));
        }
    }

    public void dispose() {
        super.dispose();

        this.getSite().getWorkbenchWindow().getSelectionService().removeSelectionListener(this);
    }

    /** 设置数据 */
    public void setData(List data) {
        this.data = data;
        viewer.setInput(this.data);
        viewer.refresh();
    }

    /** 更新数据 */
    public void refreshData() {
        data = contactService.getAllContacts();
        viewer.setInput(data);
        viewer.refresh();
    }
}

```

其中实现单击客户时，联系人列表实现联动的功能，主要是通过注册 `ISelectionService` 来实现的。

25.7.5 快速新建联系人视图

快速新建联系人与快速新建客户视图的实现方式差不多，只是这里新建的是联系人记录。快速新建联系人视图的界面效果如图 25.21 所示。

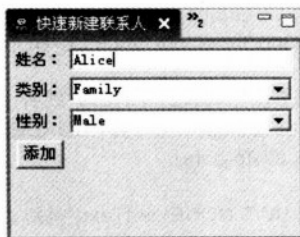


图 25.21 快速新建联系人视图界面效果图

快速新建联系人视图 QuickNewContactView 类的具体代码如下：

QuickNewContactView.java

```
package superCRM.views;

import org.eclipse.swt.SWT;
import org.eclipse.swt.events.SelectionAdapter;
import org.eclipse.swt.events.SelectionEvent;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Combo;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.Text;
import org.eclipse.ui.IViewPart;
import org.eclipse.ui.PartInitException;
import org.eclipse.ui.part.ViewPart;

import superCRM.model.IContactService;
import superCRM.model.SuperFactory;
import superCRM.pojos.ContactEO;

public class QuickNewContactView extends ViewPart {
    /** 该视图的 ID */
    public static final String ID = "superCRM.views.QuickNewContactView";
    /** 姓名 */
    private Text displayName;
    /** 类别 */
    private Combo category;
    /** 性别 */
    private Combo sex;
    public QuickNewContactView() {
        super();
    }

    public void createPartControl(Composite parent) {
        /** 设置面板布局 */
        Composite composite = new Composite(parent, SWT.NONE);
        composite.setLayout(new GridLayout(2, false));
        GridData data = new GridData(GridData.FILL_HORIZONTAL);
        /** 姓名 */
        new Label(composite, SWT.NONE).setText("姓名: ");
        displayName = new Text(composite, SWT.BORDER);
        displayName.setLayoutData(data);
        /** 类别 */
        new Label(composite, SWT.NONE).setText("类别: ");
        category = new Combo(composite, SWT.BORDER);
        category.setItems(new String[] { "Friend", "Family", "Contact" });
    }
}
```

```

category.select(0);
data = new GridData(GridData.FILL_HORIZONTAL);
category.setLayoutData(data);
/** 性别 */
new Label(composite, SWT.NONE).setText("性别: ");
sex = new Combo(composite, SWT.BORDER);
sex.setItems(new String[] { "Female", "Male" });
sex.select(0);
data = new GridData(GridData.FILL_HORIZONTAL);
sex.setLayoutData(data);
/** 添加按钮 */
Button btAdd = new Button(composite, SWT.PUSH);
btAdd.setText("添加");
btAdd.addSelectionListener(new SelectionAdapter() {

    public void widgetSelected(SelectionEvent e) {
        /** 创建联系人对象 */
        ContactEO contact = new ContactEO();
        /** 设置输入的联系人信息 */
        contact.setDisplayName(displayName.getText());
        contact.setCategory(category.getText());
        contact.setSex(sex.getText());
        /** 调用业务层保存到数据库 */
        IContactService contactService = SuperFactory.getSuperApplication().get
ContactService();
        contactService.addContact(contact);
        /** 查找当前页面是否打开了联系人列表视图 */
        IViewPart view = getViewSite().getPage().findView(ContactSummaryView.ID);
        /** 如果未打开联系人列表视图, 则打开视图 */
        if (view == null) {
            try {
                view = getViewSite().getPage().showView(ContactSummaryView.ID);
            } catch (PartInitException ee) {
                ee.printStackTrace();
            }
        }
        /** 更新联系人列表视图 */
        ContactSummaryView contactSummaryView = (ContactSummaryView) view;
        contactSummaryView.refreshData();
    }
});
}
public void setFocus() {
    displayName.setFocus();
}
}

```

25.7.6 搜索视图

搜索视图用于搜索某些关键字的客户或联系人信息，如图 25.22 所示为搜索视图的界面效果图。当单击“搜索”按钮后，将显示结果分别显示在客户列表视图或联系人列表视图中。

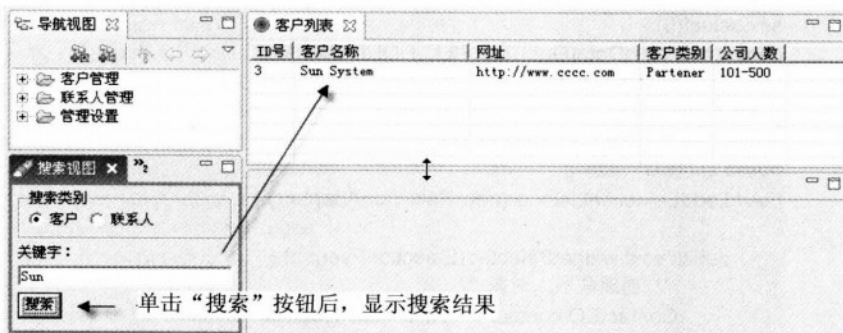


图 25.22 搜索视图的界面效果

SearchView.java

```
package superCRM.views;

import java.util.List;

import org.eclipse.jface.preference.IPreferenceStore;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.SelectionAdapter;
import org.eclipse.swt.events.SelectionEvent;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Group;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.Text;
import org.eclipse.ui.IViewPart;
import org.eclipse.ui.PartInitException;
import org.eclipse.ui.part.ViewPart;

import superCRM.SuperCRMPlugin;
import superCRM.preferences.PreferenceConstants;
import superCRM.model.IContactService;
import superCRM.model.ICustomerService;
import superCRM.model.SuperFactory;

public class SearchView extends ViewPart {
    /** 该视图的 ID */

```

```

public static final String ID = "superCRM.views.SearchView";
/** “客户”单选按钮 */
private Button btCustomer;
/** “联系人”单选按钮 */
private Button btContact;
/** 关键字文本 */
private Text keywords;
public SearchView() {
    super();
}

public void createPartControl(Composite parent) {
    /** 设置面板布局 */
    Composite composite = new Composite(parent, SWT.NONE);
    composite.setLayout(new GridLayout(1, false));
    GridData data = new GridData(GridData.FILL_HORIZONTAL);
    /** 创建分组面板放置两个单选按钮 */
    Group group = new Group(composite, SWT.NONE);
    group.setText("搜索类别");
    group.setLayout(new GridLayout(2, false));
    group.setLayoutData(data);
    /** “客户”单选按钮 */
    btCustomer = new Button(group, SWT.RADIO);
    btCustomer.setText("客户");
    /** “联系人”单选按钮 */
    btContact = new Button(group, SWT.RADIO);
    btContact.setText("联系人");
    /** 根据首选项设置默认选中“客户”还是选中“联系人” */
    IPreferenceStore store = SuperCRMPlugin.getDefault().getPreferenceStore();
    String type = store.getString(PreferenceConstants.P_DEFAULT_SEARCH);
    if (type.equals("Customer"))
        btCustomer.setSelection(true);
    else
        btContact.setSelection(true);
    /** 关键字 */
    new Label(composite, SWT.NONE).setText("关键字: ");
    keywords = new Text(composite, SWT.BORDER);
    data = new GridData(GridData.FILL_HORIZONTAL);
    keywords.setLayoutData(data);
    /** 搜索按钮 */
    Button btSearch = new Button(composite, SWT.PUSH);
    btSearch.setText("搜索");
    /** 当选中单选按钮时 */
    btSearch.addSelectionListener(new SelectionAdapter() {
        public void widgetSelected(SelectionEvent e) {
            doSearch();
        }
    });
}

```



```

/** 执行搜索 */
protected void doSearch() {
    /** 如果选中了客户 */
    if (btCustomer.getSelection()) {
        /** 获得客户服务对象 */
        ICustomerService customerService = SuperFactory.getSuperApplication().get
CustomerService();
        /** 获得查询结果的客户数据 */
        List customers = customerService.getCustomers(keywords.getText());
        /** 打开客户列表视图，并刷新数据 */
        IViewPart view = getViewSite().getPage().findView(CustomerSummaryView.ID);
        if (view == null) {
            try {
                view = getViewSite().getPage().showView(CustomerSummaryView.ID);
            } catch (PartInitException e) {
                e.printStackTrace();
            }
        }
        CustomerSummaryView customerSummaryView = (CustomerSummaryView) view;
        customerSummaryView.setData(customers);
        /** 如果选中了联系人 */
    } else if (btContact.getSelection()) {
        /** 获得联系人服务对象 */
        IContactService contactService = SuperFactory.getSuperApplication().get Contact
Service();
        /** 获得查询结果的联系人数据 */
        List contacts = contactService.getContacts(keywords.getText());
        /** 打开联系人列表视图，并刷新数据 */
        IViewPart view = getViewSite().getPage().findView(ContactSummaryView.ID);
        if (view == null) {
            try {
                view = getViewSite().getPage().showView(ContactSummaryView.ID);
            } catch (PartInitException e) {
                e.printStackTrace();
            }
        }
        ContactSummaryView contactSummaryView = (ContactSummaryView) view;
        contactSummaryView.setData(contacts);
    }
}

public void setFocus() {
    keywords.setFocus();
}
}

```


25.7.7 导航视图

导航视图是整个系统的菜单，通过树形的导航栏来实现，如图 25.23 和图 25.24 所示分别为导航视图中的导航栏展开时的效果图和右击时弹出的快捷菜单效果图。

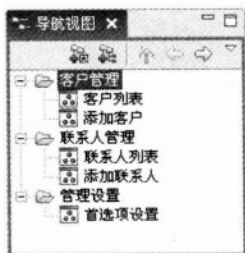


图 25.23 导航视图效果图



图 25.24 快捷菜单

导航视图 NavView 的具体代码如下：

NavView.java

```
package superCRM.views;

import java.util.ArrayList;

import org.eclipse.core.runtime.IAdaptable;
import org.eclipse.jface.action.Action;
import org.eclipse.jface.action.IMenuListener;
import org.eclipse.jface.action.IMenuManager;
import org.eclipse.jface.action.IToolBarManager;
import org.eclipse.jface.action.MenuManager;
import org.eclipse.jface.action.Separator;
import org.eclipse.jface.viewers.DoubleClickEvent;
import org.eclipse.jface.viewers.IDoubleClickListener;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.viewers.IStructuredContentProvider;
import org.eclipse.jface.viewers.IStructuredSelection;
import org.eclipse.jface.viewers.ITreeContentProvider;
import org.eclipse.jface.viewers.LabelProvider;
import org.eclipse.jface.viewers.TreeViewer;
import org.eclipse.jface.viewers.Viewer;
import org.eclipse.swt.SWT;
import org.eclipse.swt.graphics.Image;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Menu;
import org.eclipse.ui.IActionBars;
import org.eclipse.ui.ISharedImages;
import org.eclipse.ui.IWorkbenchActionConstants;
import org.eclipse.ui.PlatformUI;
```

```
import org.eclipse.ui.actions.ActionFactory;
import org.eclipse.ui.part.DrillDownAdapter;
import org.eclipse.ui.part.ViewPart;

import superCRM.SuperCRMPlugin;
import superCRM.action.ActionManager;

/** 导航菜单视图 */
public class NavView extends ViewPart {

    /** 导航的树对象 */
    private TreeViewer viewer;
    /** 可以为树添加返回导航操作对象 */
    private DrillDownAdapter drillDownAdapter;
    /** 折叠树操作 */
    private Action collapseAction;
    /** 展开树操作 */
    private Action expandAction;
    /** 双击树节点时操作 */
    private Action doubleClickAction;
    /** 该视图的 ID */
    public static final String ID = "superCRM.views.NavView";
    /** 树节点类 */
    class TreeObject implements IAdaptable {
        private String name;
        private String key;
        private TreeParent parent;
        public TreeObject(String name, String key) {
            this.name = name;
            this.key = key;
        }

        public String getName() {
            return name;
        }

        public void setParent(TreeParent parent) {
            this.parent = parent;
        }

        public TreeParent getParent() {
            return parent;
        }

        public String toString() {
            return getName();
        }

        public Object getAdapter(Class key) {
            return null;
        }

        public String getKey() {
            return key;
        }
    }
}
```

```
    }
    public void setKey(String key) {
        this.key = key;
    }
}
/** 树的父节点类 */
class TreeParent extends TreeObject {
    private ArrayList children;
    public TreeParent(String name) {
        super(name, "__Parent");
        children = new ArrayList();
    }
    public void addChild(TreeObject child) {
        children.add(child);
        child.setParent(this);
    }
    public void removeChild(TreeObject child) {
        children.remove(child);
        child.setParent(null);
    }
    public TreeObject[] getChildren() {
        return (TreeObject[]) children.toArray(new TreeObject[children.size()]);
    }
    public boolean hasChildren() {
        return children.size() > 0;
    }
}
/** 树的内容器 */
class ViewContentProvider implements IStructuredContentProvider, ITreeContentProvider {
    private TreeParent invisibleRoot;

    public void inputChanged(Viewer v, Object oldInput, Object newInput) {
    }
    public void dispose() {
    }
    public Object[] getElements(Object parent) {
        if (parent.equals(getViewSite())) {
            if (invisibleRoot == null)
                initialize();
            return getChildren(invisibleRoot);
        }
        return getChildren(parent);
    }
    public Object getParent(Object child) {
        if (child instanceof TreeObject) {
            return ((TreeObject) child).getParent();
        }
        return null;
    }
}
```

```

public Object[] getChildren(Object parent) {
    if (parent instanceof TreeParent) {
        return ((TreeParent) parent).getChildren();
    }
    return new Object[0];
}

public boolean hasChildren(Object parent) {
    if (parent instanceof TreeParent)
        return ((TreeParent) parent).hasChildren();
    return false;
}

/** 初始化树中的数据 */
private void initialize() {
    TreeObject c1 = new TreeObject("客户列表", "CUSTOMER_LIST");
    TreeObject c2 = new TreeObject("添加客户", "CUSTOMER_ADD");
    TreeParent t1 = new TreeParent("客户管理");
    t1.addChild(c1);
    t1.addChild(c2);

    TreeObject l1 = new TreeObject("联系人列表", "CONTACT_LIST");
    TreeObject l2 = new TreeObject("添加联系人", "CONTACT_ADD");
    TreeParent t2 = new TreeParent("联系人管理");
    t2.addChild(l1);
    t2.addChild(l2);

    TreeObject p2 = new TreeObject("首选项设置", "PREF");
    TreeParent t3 = new TreeParent("管理设置");
    t3.addChild(p2);

    invisibleRoot = new TreeParent("");
    invisibleRoot.addChild(t1);
    invisibleRoot.addChild(t2);
    invisibleRoot.addChild(t3);
}

/** 树的内容器 */
class ViewLabelProvider extends LabelProvider {
    public String getText(Object obj) {
        return obj.toString();
    }

    public Image getImage(Object obj) {
        String imageKey = ISharedImages.IMG_OBJ_ELEMENT;
        if (obj instanceof TreeParent)
            imageKey = ISharedImages.IMG_OBJ_FOLDER;
        return PlatformUI.getWorkbench().getSharedImages().getImage(imageKey);
    }
}

/** 构造方法 */
public NavView() {

```



```
}  
public void createPartControl(Composite parent) {  
    /** 创建树 */  
    viewer = new TreeViewer(parent, SWT.MULTI | SWT.H_SCROLL | SWT.V_SCROLL);  
    /** 初始化树 */  
    drillDownAdapter = new DrillDownAdapter(viewer);  
    viewer.setContentProvider(new ViewContentProvider());  
    viewer.setLabelProvider(new ViewLabelProvider());  
    viewer.setInput(getViewSite());  
    /** 创建树中所使用操作对象 */  
    makeActions();  
    /** 添加上下文菜单 */  
    hookContextMenu();  
    /** 添加双击事件 */  
    hookDoubleClickAction();  
    /** 添加到操作条中 */  
    contributeToActionBars();  
}  
/** 添加上下文菜单 */  
private void hookContextMenu() {  
    MenuManager menuMgr = new MenuManager("#PopupMenu");  
    menuMgr.setRemoveAllWhenShown(true);  
    menuMgr.addMenuListener(new IMenuListener() {  
        public void menuAboutToShow(IMenuManager manager) {  
            NavView.this.fillContextMenu(manager);  
        }  
    });  
    Menu menu = menuMgr.createContextMenu(viewer.getControl());  
    viewer.getControl().setMenu(menu);  
    getViewSite().registerContextMenu(menuMgr, viewer);  
}  
/** 添加到操作条中 */  
private void contributeToActionBars() {  
    IActionBars bars = getViewSite().getActionBars();  
    fillLocalPullDown(bars.getMenuManager());  
    fillLocalToolBar(bars.getToolBarManager());  
}  
/** 添加下拉菜单项 */  
private void fillLocalPullDown(IMenuManager manager) {  
    manager.add(collapseAction);  
    manager.add(new Separator());  
    manager.add(expandAction);  
}  
/** 添加上下文菜单项 */  
private void fillContextMenu(IMenuManager manager) {  
    manager.add(collapseAction);  
    manager.add(expandAction);  
    manager.add(new Separator());  
    drillDownAdapter.addNavigationActions(manager);  
}
```



```

        manager.add(new Separator(IWorkbenchActionConstants.MB_ADDITIONS));
    }
    /** 添加工具栏工具按钮 */
    private void fillLocalToolBar(IToolBarManager manager) {
        manager.add(collapseAction);
        manager.add(expandAction);
        manager.add(new Separator());
        drillDownAdapter.addNavigationActions(manager);
    }
    /** 创建视图所使用的操作对象 */
    private void makeActions() {
        /** 展开全部按钮 */
        collapseAction = new Action() {
            public void run() {
                viewer.collapseAll();
            }
        };
        collapseAction.setText("折叠全部");
        collapseAction.setToolTipText("折叠全部");

        collapseAction.setImageDescriptor(SuperCRMPlugin.getImageDescriptor("icons/collapse.gif"));
        /** 折叠全部按钮 */
        expandAction = new Action() {
            public void run() {
                viewer.expandAll();
            }
        };
        expandAction.setText("展开全部");
        expandAction.setToolTipText("展开全部");

        expandAction.setImageDescriptor(SuperCRMPlugin.getImageDescriptor("icons/expand.gif"));
        /** 双击操作 */
        doubleClickAction = new Action() {
            public void run() {
                /** 获得当前选中的树节点 */
                ISelection selection = viewer.getSelection();
                Object obj = ((IStructuredSelection) selection).getFirstElement();
                /** 如果选中的为 TreeParent 对象, 则返回 */
                if (obj instanceof TreeParent)
                    return;
                /** 如果选中的为子节点, 则根据 key 值打开相应的操作 */
                TreeObject object = (TreeObject) obj;
                if (object.getKey().equals("CUSTOMER_LIST")) {
                    ActionManager.createShowViewAction(getSite().getWorkbenchWindow(),
CustomerSummaryView.ID).run();
                } else if (object.getKey().equals("CUSTOMER_ADD")) {
                    ActionManager.createNewCustomerAction(getSite().getWorkbenchWindow()).run();
                } else if (object.getKey().equals("CONTACT_LIST")) {

```

```

        ActionManager.createShowViewAction(getSite().getWorkbenchWindow(),
ContactSummaryView.ID).run();
    } else if (object.getKey().equals("CONTACT_ADD")) {

        ActionManager.createNewContactAction(getSite().getWorkbenchWindow()).run();
    } else if (object.getKey().equals("PREF")) {

        ActionFactory.PREFERENCES.create(getSite().getWorkbenchWindow()).run();
    }
}
};
}
/** 双击树事件 */
private void hookDoubleClickAction() {
    viewer.addClickListener(new IDoubleClickListener() {
        public void doubleClick(DoubleClickEvent event) {
            doubleClickAction.run();
        }
    });
}
public void setFocus() {
    viewer.getControl().setFocus();
}
}
}

```

25.8 新建客户联系人向导

虽然提供了快速新建客户和快速新建联系人的视图，但在实际的应用中，通常新建时要输入新建客户和联系人更多的信息，并且使用向导来提示用户输入不同信息。

25.8.1 新建客户向导

在新建客户时，通常在创建完新客户后，用户会直接创建该客户所属的联系人。此时就需要向导创建客户。如图 25.25 所示为新建客户向导的第一个页面“新建客户”的界面效果。如图 25.26 所示为新建该客户联系人信息的页面。

按照两个页面的设置完成后，单击“完成”按钮，不仅新建了一个客户记录，而且为该客户新建了一个联系人，如图 25.27 所示。

新建客户向导主要涉及 3 个类。

- ❑ NewCustomerWizard: 新建客户向导类。
- ❑ NewCustomerWizardPage: 新建客户页面。
- ❑ NewContactWizardPage: 新建联系人页面。

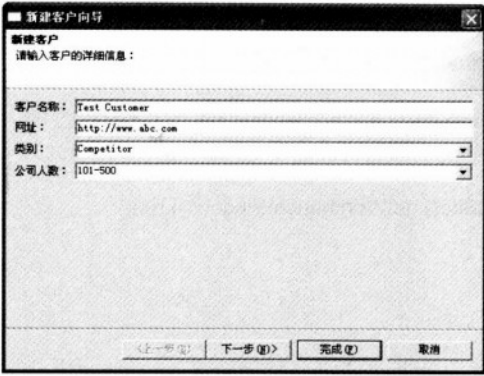


图 25.25 “新建客户”页面

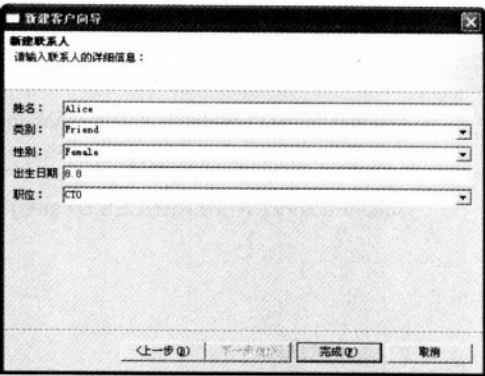


图 25.26 “新建联系人”页面

客户列表				
ID号	客户名称	网址	客户类别	公司人数
1	NestSafe	http://www.aaaa.com	Customer	51-100
2	Apache Foundation	http://www.bbbb.com	Customer	11-50
3	Sun System	http://www.cccc.com	Partner	101-500
4	Test Customer	http://www.abc.com	Competitor	101-500

联系人列表						
ID号	姓名	客户名称	类别	性别	出生日期	职位
4	Alice	Test Customer	Friend	Female	8.8	CTO

图 25.27 使用客户向导创建客户后的效果

以下是这 3 个类具体实现的代码：

NewCustomerWizard.java

```
package superCRM.dialog;

import org.eclipse.jface.wizard.Wizard;
import org.eclipse.ui.IWorkbenchWindow;

import superCRM.model.IContactService;
import superCRM.model.ICustomerService;
import superCRM.model.SuperFactory;
import superCRM.pojos.ContactEO;
import superCRM.pojos.CustomerEO;

public class NewCustomerWizard extends Wizard {
    /** 新建客户页面 */
    private NewCustomerWizardPage newCustomerPage;

    /** 新建联系人页面 */
    private NewContactWizardPage newContactPage;
```

```

/** 构造方法，初始化页面 */
public NewCustomerWizard(IWorkbenchWindow window) {
    newCustomerPage = new NewCustomerWizardPage();
    this.addPage(newCustomerPage);
    newContactPage = new NewContactWizardPage();
    this.addPage(newContactPage);
    this.setWindowTitle("新建客户向导");
}

/** 单击“完成”按钮时 */
public boolean performFinish() {
    /** 获得新建客户页面输入的客户对象 */
    CustomerEO customer = newCustomerPage.getCustomer();
    /** 如果客户对象不为 null */
    if (customer != null) {
        /** 调用业务层，保存客户到数据库中 */
        ICustomerService customerService = SuperFactory.getSuperApplication().getCustomerService();
        CustomerEO c = customerService.addCustomer(customer);
        /** 获得新建联系人页面的联系人对象 */
        ContactEO contact = newContactPage.getContact();
        /** 如果对象为 null 或客户对象为 null */
        if (c == null || contact == null)
            return true;
        /** 调用业务层，保存该联系人到数据库 */
        IContactService contactService = SuperFactory.getSuperApplication().getContactService();
        contact.setCustomerId(c.getId());
        contactService.addContact(contact);
    }
    return true;
}
}

```

NewCustomerWizardPage.java

```

package superCRM.dialog;

import org.eclipse.jface.wizard.WizardPage;
import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Combo;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.Text;

import superCRM.pojos.CustomerEO;

```



```
;  
  
public class NewCustomerWizardPage extends WizardPage {  
    /** 页面名称 */  
    public static final String NEW_CUSTOMER_PAGE = "CUSTOMER";  
    /** 客户名称 */  
    private Text displayName;  
    /** 网址 */  
    private Text website;  
    /** 客户分类 */  
    private Combo category;  
    /** 公司人数 */  
    private Combo number;  
    private CustomerEO customer;  
    public NewCustomerWizardPage() {  
        super(NEW_CUSTOMER_PAGE, "新建客户", null);  
        this.setTitle("新建客户");  
        this.setDescription("请输入客户的详细信息: ");  
    }  
  
    public void createControl(Composite parent) {  
        /** 设置面板布局 */  
        Composite composite = new Composite(parent, SWT.NONE);  
        composite.setLayout(new GridLayout(2, false));  
        GridData data = new GridData(GridData.FILL_HORIZONTAL);  
        /** 客户名称 */  
        new Label(composite, SWT.NONE).setText("客户名称: ");  
        displayName = new Text(composite, SWT.BORDER);  
        displayName.setLayoutData(data);  
        /** 网址 */  
        new Label(composite, SWT.NONE).setText("网址: ");  
        website = new Text(composite, SWT.BORDER);  
        data = new GridData(GridData.FILL_HORIZONTAL);  
        website.setLayoutData(data);  
        /** 客户分类 */  
        new Label(composite, SWT.NONE).setText("类别: ");  
        category = new Combo(composite, SWT.BORDER);  
        category.setItems(new String[] { "Customer", "Partener", "Competitor" });  
        category.select(0);  
        data = new GridData(GridData.FILL_HORIZONTAL);  
        category.setLayoutData(data);  
        /** 公司人数 */  
        new Label(composite, SWT.NONE).setText("公司人数: ");  
        number = new Combo(composite, SWT.BORDER);  
        number.setItems(new String[] { "1-10", "11-50", "51-100", "101-500", "500-" });  
        number.select(0);  
        data = new GridData(GridData.FILL_HORIZONTAL);  
        number.setLayoutData(data);  
        setControl(composite);  
    }  
}
```



```
}

/** 获得用户输入的客户信息 */
public CustomerEO getCustomer() {
    if (!displayName.getText().equals(" ")) {
        customer = new CustomerEO();
        customer.setDisplayName(displayName.getText());
        customer.setWebSite(website.getText());
        customer.setCategory(category.getText());
        customer.setNumberEmployee(number.getText());
    }
    return customer;
}
}
```

NewContactWizardPage.java

```
package superCRM.dialog;

import org.eclipse.jface.wizard.WizardPage;
import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Combo;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.Text;

import superCRM.pojos.ContactEO;

public class NewContactWizardPage extends WizardPage {
    /** 页面名称 */
    public static final String NEW_CONTACT_PAGE = "CONTACT";
    /** 姓名 */
    private Text displayName;
    /** 类别 */
    private Combo category;
    /** 性别 */
    private Combo sex;
    /** 出生日期 */
    private Text birthDate;
    /** 职位 */
    private Combo jobTitle;
    /** 联系人对象 */
    private ContactEO contact;
    public NewContactWizardPage() {
        super(NEW_CONTACT_PAGE, "新建联系人", null);
        this.setTitle("新建联系人");
        this.setDescription("请输入联系人的详细信息:");
    }
}
```

```

}
public void createControl(Composite parent) {
    /** 设置面板布局 */
    Composite composite = new Composite(parent, SWT.NONE);
    composite.setLayout(new GridLayout(2, false));
    GridData data = new GridData(GridData.FILL_HORIZONTAL);
    /** 姓名 */
    new Label(composite, SWT.NONE).setText("姓名: ");
    displayName = new Text(composite, SWT.BORDER);
    displayName.setLayoutData(data);
    /** 类别 */
    new Label(composite, SWT.NONE).setText("类别: ");
    category = new Combo(composite, SWT.BORDER);
    category.setItems(new String[] { "Friend", "Family", "Contact" });
    category.select(0);
    data = new GridData(GridData.FILL_HORIZONTAL);
    category.setLayoutData(data);
    /** 性别 */
    new Label(composite, SWT.NONE).setText("性别: ");
    sex = new Combo(composite, SWT.BORDER);
    sex.setItems(new String[] { "Female", "Male" });
    sex.select(0);
    data = new GridData(GridData.FILL_HORIZONTAL);
    sex.setLayoutData(data);
    /** 出生日期 */
    new Label(composite, SWT.NONE).setText("出生日期:");
    birthDate = new Text(composite, SWT.BORDER);
    data = new GridData(GridData.FILL_HORIZONTAL);
    birthDate.setLayoutData(data);
    /** 职位: */
    new Label(composite, SWT.NONE).setText("职位: ");
    jobTitle = new Combo(composite, SWT.BORDER);
    jobTitle.setItems(new String[] { "CEO", "CTO", "CFO" });
    jobTitle.select(0);
    data = new GridData(GridData.FILL_HORIZONTAL);
    jobTitle.setLayoutData(data);
    setControl(composite);
}
/** 获得用户输入的联系人信息 */
public ContactEO getContact() {
    if (!displayName.getText().equals("")) {
        contact = new ContactEO();
        contact.setDisplayName(displayName.getText());
        contact.setCategory(category.getText());
        contact.setSex(sex.getText());
        contact.setBirthDate(birthDate.getText());
        contact.setJobTitle(jobTitle.getText());
    }
    return contact;
}

```

```

}
}

```

25.8.2 新建联系人向导

与新建客户向导不同,当新建联系人时,需要选择该联系人所属的客户,然后再输入联系人信息。如图 25.28 所示为选择客户页面的效果图,如图 25.29 所示为选择客户页面后输入联系人的页面效果图。

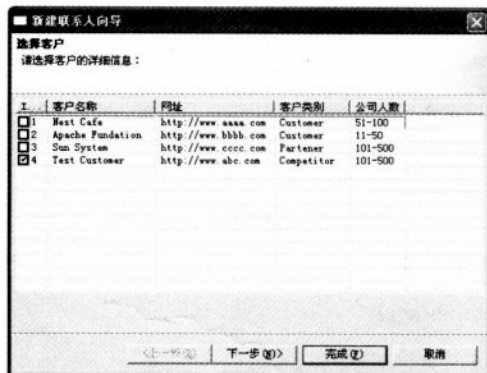


图 25.28 “选择客户”页面

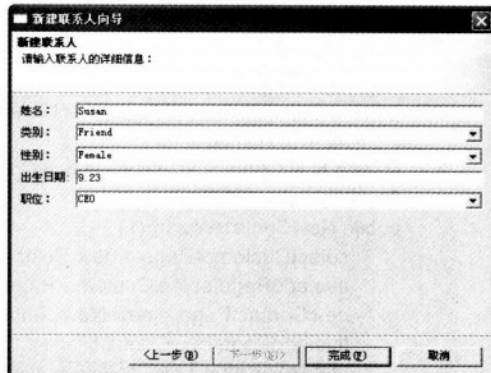


图 25.29 “输入联系人”页面

按照两个页面的设置完成后,单击“完成”按钮,即为第一个页面中选中的客户新建了一个联系人,如图 25.30 所示。

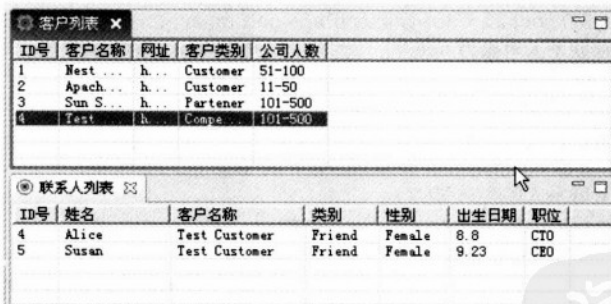


图 25.30 新建联系人向导创建的联系人后的结果示意图

新建联系人向导主要也涉及 3 个类。

- ☐ NewContactWizard: 新建联系人向导类。
- ☐ SelectCustomerPage: 选择客户页面。
- ☐ NewContactWizardPage: 新建联系人页面。

其中, NewContactWizardPage 页面上文已经提到过,这里不多作介绍。以下是其他两个类具体实现的代码:

NewContactWizard.java

```

package superCRM.dialog;

import org.eclipse.jface.wizard.Wizard;

import superCRM.model.IContactService;
import superCRM.model.SuperFactory;
import superCRM.pojos.ContactEO;
import superCRM.pojos.CustomerEO;

public class NewContactWizard extends Wizard {
    /** 选择客户页面 */
    private SelectCustomerPage selectCustomerPage;
    /** 新建联系人页面 */
    private NewContactWizardPage newContactPage;
    /** 构造方法, 初始化页面 */
    public NewContactWizard() {
        selectCustomerPage = new SelectCustomerPage();
        this.addPage(selectCustomerPage);
        newContactPage = new NewContactWizardPage();
        this.addPage(newContactPage);
        this.setWindowTitle("新建联系人向导");
    }

    /** 单击“完成”按钮时 */
    public boolean performFinish() {
        /** 获得新建联系人页面的联系人对象 */
        ContactEO contact = newContactPage.getContact();
        /** 如果联系人对象为 null */
        if (contact == null)
            return true;
        /** 获得选择客户页面所选择的客户对象 */
        CustomerEO customer = selectCustomerPage.getCustomer();
        /** 设置联系人的客户 ID */
        contact.setCustomerId(customer.getId());
        /** 调用业务层, 保存联系人信息 */
        IContactService contactService = SuperFactory.getSuperApplication().getContactService();
        contactService.addContact(contact);
        return true;
    }
}

```

SelectCustomerPage.java

```

package superCRM.dialog;

import java.util.List;

```



```

import org.eclipse.jface.viewers.CheckboxTableViewer;
import org.eclipse.jface.viewers.ISelectionChangedListener;
import org.eclipse.jface.viewers.SelectionChangedEvent;
import org.eclipse.jface.wizard.WizardPage;
import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.FillLayout;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.TableColumn;

import superCRM.model.ICustomerService;
import superCRM.model.SuperFactory;
import superCRM.pojos.CustomerEO;
import superCRM.table.TableContentProvider;
import superCRM.table.TableLabelProvider;

public class SelectCustomerPage extends WizardPage {
    /** 页面名称 */
    public static final String SELECT_CUSTOMER_PAGE = "SELECT_CUSTOMER";
    /** 客户列表的标题 */
    public static final String[] COLUMN_NAME = { "ID 号", "客户名称", "网址", "客户类别", "公司
人数" };
    /** 带选择框的表格 */
    private CheckboxTableViewer viewer;
    /** 表格数据 */
    public List data;
    /** 客户服务对象 */
    private ICustomerService customerService;
    /** 客户对象 */
    private CustomerEO cusotmer;
    public SelectCustomerPage() {
        super(SELECT_CUSTOMER_PAGE, "新建客户", null);
        this.setTitle("选择客户");
        this.setDescription("请选择客户的详细信息: ");
        this.setPageComplete(false);
        customerService = SuperFactory.getSuperApplication().getCustomerService();
        data = customerService.getAllCustomers();
    }

    public void createControl(Composite parent) {
        /** 设置面板布局 */
        Composite composite = new Composite(parent, SWT.NONE);
        composite.setLayout(new FillLayout());
        /** 初始化表数据 */
        viewer = CheckboxTableViewer.newCheckList(composite, SWT.FULL_SELECTION |
SWT.SINGLE);
        for (int i = 0; i < COLUMN_NAME.length; i++) {
            new TableColumn(viewer.getTable(), SWT.LEFT).setText(COLUMN_NAME[i]);
            viewer.getTable().getColumn(i).pack();
        }
    }
}

```



```

    }
    viewer.getTable().setHeaderVisible(true);
    viewer.getTable().setLinesVisible(true);
    viewer.setContentProvider(new TableContentProvider());
    viewer.setLabelProvider(new TableLabelProvider());
    viewer.setInput(data);
    /** 注册表格单击监听器 */
    viewer.addSelectionChangedListener(new ISelectionChangedListener() {
        /** 当单击表格中一行时 */
        public void selectionChanged(SelectionChangedEvent event) {
            /** 获得表格选中的行 */
            Object[] checkedObj = viewer.getCheckedElements();
            /** 如果不是选择了一个客户，则提示错误 */
            if (checkedObj.length != 1) {
                setErrorMessage("请选择一个客户");
                setPageComplete(false);
                return;
            }
            /** 设置下一步可用状态 */
            setErrorMessage(null);
            setPageComplete(true);
            /** 将选中的客户对象赋值给该页面的客户对象 */
            cusotmer = (CustomerEO) checkedObj[0];
        }
    });
    setControl(composite);
}

/** 获得该页面的客户对象 */
public CustomerEO getCusotmer() {
    return cusotmer;
}

/** 设置该页面的客户对象 */
public void setCusotmer(CustomerEO cusotmer) {
    this.cusotmer = cusotmer;
}
}

```

25.9 首选项的实现

首选项是系统中不可缺少的一个功能，用户的一些喜好都可以设置在首选项中，该系统中所涉及首选项的类如下：

- PreferenceInitializer：系统装载后，首选项默认的值。

- ❑ PreferenceConstants: 保存了首选项 key 值的常量。
- ❑ BasicPreferencePage: 基本设置页面。
- ❑ LoginPreferencePage: 登录设置页面。

PreferenceInitializer 类继承自 AbstractPreferenceInitializer 类, 并实现了 initializeDefaultPreferences 方法。当系统装载首选项时, 首先调用该方法设置首选项的默认值。该类的代码如下:

PreferenceInitializer.java

```
package superCRM.preferences;

import org.eclipse.core.runtime.preferences.AbstractPreferenceInitializer;
import org.eclipse.jface.preference.IPreferenceStore;

import superCRM.SuperCRMPlugin;

public class PreferenceInitializer extends AbstractPreferenceInitializer {
    /** 设置首选项默认值 */
    public void initializeDefaultPreferences() {
        IPreferenceStore store = SuperCRMPlugin.getDefault().getPreferenceStore();
        store.setDefault(PreferenceConstants.P_AUTO_LOGIN, false);
        store.setDefault(PreferenceConstants.P_USER_NAME, "admin");
        store.setDefault(PreferenceConstants.P_PASSWORD, "admin");
        store.setDefault(PreferenceConstants.P_DEFAULT_SEARCH, "Customer");
    }
}
```

PreferenceConstants 类保存了首选项 key 值的常量, 代码如下:

PreferenceConstants.java

```
package superCRM.preferences;

public class PreferenceConstants {
    /** 是否自动登录 */
    public static final String P_AUTO_LOGIN = "autoLogin";
    /** 用户名 */
    public static final String P_USER_NAME = "userName";
    /** 密码 */
    public static final String P_PASSWORD = "password";
    /** 默认搜索类型 */
    public static final String P_DEFAULT_SEARCH = "defaultSearch";
}
```

BasicPreferencePage 类为基本设置页面, 代码运行后界面效果如图 25.31 所示。该类的具体代码如下:

BasicPreferencePage.java

```
package superCRM.preferences;
```

```

import org.eclipse.jface.preference.*;
import org.eclipse.ui.IWorkbenchPreferencePage;
import org.eclipse.ui.IWorkbench;
import superCRM.SuperCRMPlugin;

public class BasicPreferencePage extends FieldEditorPreferencePage implements IWorkbenchPreferencePage {

    public BasicPreferencePage() {
        super(GRID);
        /** 设置首选项保存对象 */
        setPreferenceStore(SuperCRMPlugin.getDefault().getPreferenceStore());
        /** 设置首选项页面描述信息 */
        setDescription("基本设置");
    }

    public void createFieldEditors() {
        /** 添加一个单选按钮组字段 */
        addField(new RadioGroupFieldEditor(PreferenceConstants.P_DEFAULT_SEARCH, "默认  
的搜索类型", 2, new String[][] { { "客户", "Customer" }, { "联系人", "Contact" } },
        getFieldEditorParent(), true));
    }

    public void init(IWorkbench workbench) {
    }
}

```

LoginPreferencePage 类为登录设置页面，代码运行后界面效果如图 25.32 所示。

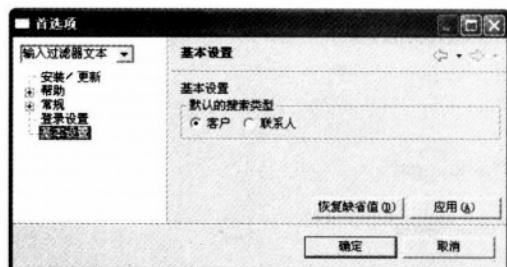


图 25.31 基本设置首选项页面

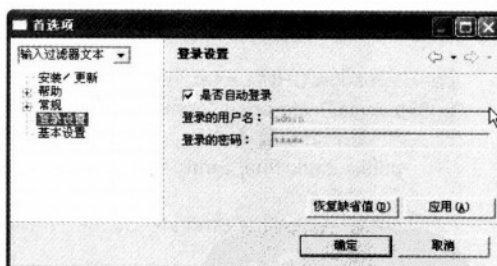


图 25.32 登录设置首选项页面

该首选项页面类的具体代码如下：

LoginPreferencePage.java

```

package superCRM.preferences;

import org.eclipse.jface.preference.IPreferenceStore;
import org.eclipse.jface.preference.PreferencePage;
import org.eclipse.swt.SWT;

```

```
import org.eclipse.swt.events.SelectionAdapter;
import org.eclipse.swt.events.SelectionEvent;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Control;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.Text;
import org.eclipse.ui.IWorkbench;
import org.eclipse.ui.IWorkbenchPreferencePage;

import superCRM.SuperCRMPlugin;

public class LoginPreferencePage extends PreferencePage implements IWorkbenchPreferencePage {
    /** 是否自动登录按钮 */
    private Button bAutoLogin;
    /** 用户名 */
    private Text userName;
    /** 密码 */
    private Text password;
    public LoginPreferencePage() {
        super("登录设置");
        setPreferenceStore(SuperCRMPlugin.getDefault().getPreferenceStore());
    }

    protected Control createContents(Composite parent) {
        /** 设置面板布局 */
        Composite composite = new Composite(parent, SWT.NONE);
        GridData gridData = new GridData(GridData.FILL_BOTH);
        composite.setLayoutData(gridData);
        composite.setLayout(new GridLayout(2, false));
        /** 是否登录按钮 */
        bAutoLogin = new Button(composite, SWT.CHECK);
        bAutoLogin.setText("是否自动登录");
        /** 根据首选项设置登录按钮状态 */

        bAutoLogin.setSelection(getPreferenceStore().getBoolean(PreferenceConstants.P_AUTO_LOGIN));

        gridData = new GridData();
        gridData.horizontalSpan = 2;
        bAutoLogin.setLayoutData(gridData);
        /** 注册登录按钮选中事件监听器 */
        bAutoLogin.addSelectionListener(new SelectionAdapter() {
            /** 当选中是否登录按钮时 */
            public void widgetSelected(SelectionEvent e) {
                /** 根据登录状态设置用户名和密码框的可用状态 */
                setLoginEnabled(bAutoLogin.getSelection());
            }
        });
    }
}
```



```

    }
});
/** 用户名 */
new Label(composite, SWT.NONE).setText("登录的用户名: ");
userName = new Text(composite, SWT.BORDER);

userName.setText(getPreferenceStore().getString(PreferenceConstants.P_USER_NAME));
gridData = new GridData(GridData.FILL_HORIZONTAL);
userName.setLayoutData(gridData);
/** 密码 */
new Label(composite, SWT.NONE).setText("登录的密码: ");
password = new Text(composite, SWT.BORDER);
password.setEchoChar('*');

password.setText(getPreferenceStore().getString(PreferenceConstants.P_PASSWORD));
gridData = new GridData(GridData.FILL_HORIZONTAL);
password.setLayoutData(gridData);
/** 根据登录状态设置用户名和密码框的可用状态 */
setLoginEnabled(bAutoLogin.getSelection());
return composite;
}

public void init(IWorkbench workbench) {

}

/** 设置用户名和密码文本框状态 */
private void setLoginEnabled(boolean enable) {
    userName.setEnabled(!enable);
    password.setEnabled(!enable);
}

/** 单击“确定”按钮时，设置首选项的值 */
public boolean performOk() {
    IPreferenceStore store = getPreferenceStore();
    store.setValue(PreferenceConstants.P_AUTO_LOGIN, bAutoLogin.getSelection());
    store.setValue(PreferenceConstants.P_USER_NAME, userName.getText());
    store.setValue(PreferenceConstants.P_PASSWORD, password.getText());
    return true;
}

/** 单击“恢复缺省”按钮时，恢复默认值 */
protected void performDefaults() {
    IPreferenceStore store = getPreferenceStore();

    bAutoLogin.setSelection(store.getDefaultBoolean(PreferenceConstants.P_AUTO_LOGIN));
    userName.setText(store.getDefaultString(PreferenceConstants.P_USER_NAME));
    password.setText(store.getDefaultString(PreferenceConstants.P_PASSWORD));
    setLoginEnabled(bAutoLogin.getSelection());
}

```



```
}  
  
}
```

25.10 plugin.xml 文件清单

该 RCP 项目中，插件清单文件 plugin.xml 文件的配置代码如下：

plugin.xml

```
<?xml version="1.0" encoding="UTF-8"?>  
<?eclipse version="3.0"?>  
<plugin>  
  <extension  
    id="superCRM"  
    point="org.eclipse.core.runtime.applications">  
    <application>  
      <run  
        class="superCRM.intro.Application">  
      </run>  
    </application>  
  </extension>  
  <extension  
    point="org.eclipse.ui.perspectives">  
    <perspective  
      class="superCRM.intro.Perspective"  
      icon="icons/logo.gif"  
      id="SuperCRM.perspective"  
      name="默认透视图">  
    </perspective>  
  </extension>  
  <extension  
    point="org.eclipse.ui.intro">  
    <intro  
      class="org.eclipse.ui.intro.config.CustomizableIntroPart"  
      id="superCRM.introID">  
    </intro>  
    <introProductBinding  
      introId="superCRM.introID"  
      productId="com.fengmanfei.SuperCRM">  
    </introProductBinding>  
  </extension>  
  <extension  
    point="org.eclipse.ui.intro.config">  
    <config  
      introId="superCRM.introID"  
      content="introContent.xml"  
      id="SuperCRM.configID">
```

```
<presentation
    home-page-id="root">
    <implementation
        kind="html"
        os="win32,linux,macosx">
    </implementation>
</presentation>
</config>
</extension>
<extension
    point="org.eclipse.ui.intro.configExtension">
    <configExtension
        content="ext.xml"
        configId="SuperCRM.configId">
    </configExtension>
</extension>
<extension
    point="org.eclipse.ui.preferencePages">
    <page
        class="superCRM.preferences.BasicPreferencePage"
        id="superCRM.preferences.BasicPreferencePage"
        name="基本设置"/>
    <page
        class="superCRM.preferences.LoginPreferencePage"
        id="superCRM.preferences.LoginPreferencePage"
        name="登录设置"/>
</extension>
<extension
    point="org.eclipse.core.runtime.preferences">
    <initializer class="superCRM.preferences.PreferenceInitializer"/>
</extension>
<extension
    point="org.eclipse.ui.views">
    <view
        allowMultiple="true"
        class="superCRM.views.NavView"
        icon="icons/nav.gif"
        id="superCRM.views.NavView"
        name="导航视图"/>
    <view
        allowMultiple="false"
        class="superCRM.views.SearchView"
        icon="icons/search.gif"
        id="superCRM.views.SearchView"
        name="搜索视图"/>
    <view
        allowMultiple="false"
        class="superCRM.views.QuickNewCustomerView"
        icon="icons/customer.gif"
```

```
id="superCRM.views.QuickNewCustomerView"
name="快速新建客户"/>
<view
    allowMultiple="false"
    class="superCRM.views.QuickNewContactView"
    icon="icons/contact.gif"
    id="superCRM.views.QuickNewContactView"
    name="快速新建联系人"/>
<view
    allowMultiple="false"
    class="superCRM.views.CustomerSummaryView"
    icon="icons/list.gif"
    id="superCRM.views.CustomerSummaryView"
    name="客户列表"/>
<view
    class="superCRM.views.ContactSummaryView"
    icon="icons/list2.gif"
    id="superCRM.views.ContactSummaryView"
    name="联系人列表"/>
</extension>
<extension
    point="org.eclipse.ui.editors">
    <editor
        class="superCRM.editor.CustomerDetailEditor"
        default="false"
        icon="icons/customer.gif"
        id="superCRM.editor.CustomerDetailEditor"
        name="客户详细信息"/>
    </extension>
<extension
    point="org.eclipse.ui.commands">
    <keyBinding
        commandId="superCRM.action.NewContactAction"
        keyConfigurationId="org.eclipse.ui.defaultAcceleratorConfiguration"
        keySequence="M1+M2+P"/>
    <category
        id="superCRM.commands"
        name="superCRM"/>
    <command
        categoryId="superCRM.commands"
        id="superCRM.action.NewContactAction"
        name="新建联系人"/>
    <keyBinding
        commandId="superCRM.action.NewCustomerAction"
        keyConfigurationId="org.eclipse.ui.defaultAcceleratorConfiguration"
        keySequence="M1+M2+O"/>
    <command
        categoryId="superCRM.commands"
        id="superCRM.action.NewCustomerAction"
```

```
        name="新建客户"/>
    </extension>
    <extension
        id="superCRM"
        point="org.eclipse.core.runtime.products">
        <product
            application="com.fengmanfei.SuperCRM.superCRM"
            name="superCRM"/>
        </extension>
    </plugin>
```

25.11 本章小结

通过本章一个完整的客户关系管理系统的介绍，读者应该对开发 RCP 程序有了一个清晰的认识。该系统中用到了本书所学的所有知识，并且在整体构架上也提供了一套很好的设计思路，读者可以直接应用到实际的项目中去。

如果读者对本章的程序理解有一定的困难，请读者参阅本书中相关的章节讲解。



[General Information]

书名=ECLIPSE SWT JFACE核心应用

作者=那静编著

页数=626

SS号=11817909

出版日期=2007.03

出版社=北京市：清华大学出版社

SSLIB-JPG=http://png3.5read.com/image/ss2jpg.dll?did=b27&pid=86402BC6AB94960E0C286AAA691EE3DD208EB1A9BF663DD78A46473E7337939EA1EDBAA58113771A492BE4CF76EBE41B87F168F6DF33CAE878D777B6C84BD23D95DDE21E81A15943FAAEA9135A778A07F5B7E7460B0DBA83FBFC0BDD3CC5FC1FF8AC0ADF569268211D1BF02257AE6CAC3D1C&jid=

第1篇 SWT起步篇

第1章 Java语言的GUI历史

- 1.1 最初的AWT
- 1.2 Swing工具包
- 1.3 Eclipse的诞生
- 1.4 Eclipse贡献SWT工具包
 - 1.4.1 SWT的结构
 - 1.4.2 SWT所支持的操作系统
- 1.5 Sun AWT/Swing与Eclipse SWT
 - 1.5.1 Swing与SWT的比较
 - 1.5.2 SWT的优势和不足
- 1.6 SWT与JFace、Eclipse
 - 1.6.1 JFace是SWT的扩展
 - 1.6.2 Eclipse的UI界面基于JFace
- 1.7 本章小结

第2章 配置SVVT开发环境

- 2.1 下载和安装Eclipse
 - 2.1.1 Eclipse下载页面介绍
 - 2.1.2 下载Eclipse
 - 2.1.3 安装Eclipse语言包
 - 2.1.4 在不同的语言中切换
- 2.2 直接获取SWT工具包
- 2.3 下载和安装Visual Editor
 - 2.3.1 Visual Editor的下载
 - 2.3.2 Visual Editor的安装
- 2.4 第一个SWT程序
 - 2.4.1 创建SWT程序
 - 2.4.2 编译和运行程序
- 2.5 本章小结

第3章 Eclipse开发环境概述

- 3.1 Eclipse界面一览
- 3.2 Eclipse项目的文件结构
 - 3.2.1 设置编译后.class文件的保存目录
 - 3.2.2 导入项目使用的包
 - 3.2.3 设置编译方式
 - 3.2.4 运行程序
- 3.3 常用的代码编辑功能
 - 3.3.1 添加注释
 - 3.3.2 自定义格式化代码
 - 3.3.3 自动生成getter和setter代码
 - 3.3.4 代码的重构
 - 3.3.5 查看源代码
 - 3.3.6 代码的展开和折叠
 - 3.3.7 代码比较
 - 3.3.8 子类中覆盖父类的方法
- 3.4 代码错误提示
 - 3.4.1 如何定位错误
 - 3.4.2 自动修正错误
- 3.5 文件查找
 - 3.5.1 文件内部查找
 - 3.5.2 项目内查找
- 3.6 使用快捷键
 - 3.6.1 显示快捷键说明
 - 3.6.2 自定义快捷键
- 3.7 本章小结

第2篇 SWT进阶篇

第4章 SWT开发基础

- 4.1 SWT应用程序基本结构
- 4.2 Display类
 - 4.2.1 Display类概述
 - 4.2.2 Display类常用方法
- 4.3 Shell类
 - 4.3.1 Shell类概述
 - 4.3.2 不同窗口的样式
 - 4.3.3 应用多个样式
 - 4.3.4 Shell类的主要方法
 - 4.3.5 创建多个窗口

- 4.4 SWT包类结构
- 4.5 本章小结
- 第5章 SWT基本组件
 - 5.1 SWT控件类概述
 - 5.1.1 窗口小部件:Widget
 - 5.1.2 Widget的继承关系
 - 5.1.3 SWT中的子类
 - 5.1.4 控件(Controls)与面板(Composites)
 - 5.1.5 Widgets不是Controls
 - 5.2 按钮(Button)
 - 5.2.1 普通按钮(SWT.PUSH)
 - 5.2.2 切换按钮(SWT.TOGGLE)
 - 5.2.3 箭头按钮(SWT.ARROW)
 - 5.2.4 单选按钮(SWT.RADIO)
 - 5.2.5 多选按钮(SWT.CHECK)
 - 5.2.6 常用的方法
 - 5.3 标签(Label)
 - 5.3.1 文本标签
 - 5.3.2 分割线标签
 - 5.3.3 自定义标签(CLabel)
 - 5.4 文本框(Text)
 - 5.4.1 文本框的样式
 - 5.4.2 文本框程序示例
 - 5.4.3 常用的方法
 - 5.5 列表框(List)
 - 5.5.1 列表框的样式
 - 5.5.2 列表框程序示例
 - 5.5.3 常用的方法
 - 5.6 组合框(Combo)
 - 5.6.1 组合框的样式
 - 5.6.2 组合框程序示例
 - 5.6.3 组合框的常用方法
 - 5.6.4 自定义组合框CCombo类
 - 5.7 本章小结
- 第6章 面板容器类
 - 6.1 面板类(Composite)
 - 6.1.1 面板类的样式
 - 6.1.2 面板类的常用方法
 - 6.2 分组框(Group)
 - 6.3 选项卡(TabFolder)
 - 6.3.1 选项卡的基本构成
 - 6.3.2 设置底部显示选项卡
 - 6.3.3 设置选项卡图标
 - 6.3.4 选项卡的常用方法
 - 6.4 自定义选项卡(CTabFolder)
 - 6.4.1 带有“关闭”按钮的选项卡
 - 6.4.2 带有边框的选项卡
 - 6.4.3 显示“最大化/最小化”按钮
 - 6.4.4 设置选项卡的颜色和背景图片
 - 6.4.5 仿Eclipse编辑区的选项卡
 - 6.4.6 限制选项卡文字的长度
 - 6.4.7 设置右上角控件
 - 6.4.8 自定义选项的常用方法
 - 6.5 分割窗框(SashForm)
 - 6.5.1 分割窗框的样式
 - 6.5.2 设置窗框显示的比例
 - 6.5.3 设置窗框最大化所显示的控件
 - 6.6 自定义分割框(CBanner)
 - 6.6.1 改变分割线的外观
 - 6.6.2 Eclipse中的CBanner
 - 6.7 滚动面板(ScrolledComposite)
 - 6.7.1 设置滚动条的样式
 - 6.7.2 滚动面板的其他方法
 - 6.8 本章小结
- 第7章 SWT布局管理器
 - 7.1 布局管理器概述
 - 7.1.1 绝对定位
 - 7.1.2 托管定位
 - 7.1.3 常见的布局管理器
 - 7.2 FillLayout(充满式布局)
 - 7.2.1 水平填充(默认)和垂直填充
 - 7.2.2 设置四周补白

- 7.3 RowLayout (行列式布局)
 - 7.3.1 设置折行显示: wrap 属性
 - 7.3.2 设置空间大小: pack 属性
 - 7.3.3 设置填充方式: type 属性
 - 7.3.4 设置是否充满整行: justify 属性
 - 7.3.5 设置补白和间隔
 - 7.3.6 设置控件的大小 RowData
 - 7.3.7 设置是否等宽或等高: fill 属性
- 7.4 GridLayout (网格式布局)
 - 7.4.1 设置网格的列数: numColumns 属性
 - 7.4.2 设置网格等宽: makeColumnsEqualWidth 属性
 - 7.4.3 设置补白和间隔
 - 7.4.4 使用 GridData 对象
 - 7.4.5 设置单元对齐方式: horizontalAlignment 和 verticalAlignment 属性
 - 7.4.6 设置缩进大小: horizontalIndent 和 verticalIndent 属性
 - 7.4.7 设置单元格跨行和跨列显示: horizontalSpan 和 verticalSpan 属性
 - 7.4.8 设置单元格空间的抢占方式: grabExcessHorizontalSpace 和 grabExcessVerticalSpace 属性
 - 7.4.9 设置的控件大小: minimumWidth 和 minimumHeight 属性
 - 7.4.10 设置控件大小: widthHint 和 heightHint 属性
 - 7.4.11 样式常量对照表
- 7.5 FormLayout (表格式布局)
 - 7.5.1 设置补白和间隔
 - 7.5.2 使用 FormData 对象
 - 7.5.3 使用 FormAttachment 对象
 - 7.5.4 设置控件的相对位置
- 7.6 StackLayout (堆栈式布局)
- 7.7 自定义布局管理器
 - 7.7.1 布局的基本原理
 - 7.7.2 布局计算的常用方法
 - 7.7.3 自定义布局类 (BorderLayout)
- 7.8 使用 VE 可视化布局
 - 7.8.1 创建可视化的类
 - 7.8.2 进行布局设置
- 7.9 本章小结
- 第8章 SWT 中的事件模型
 - 8.1 事件模型概述
 - 8.1.1 监听器 (Listener)
 - 8.1.2 事件 (Event)
 - 8.1.3 注册监听器
 - 8.1.4 适配器
 - 8.1.5 常见的事件
 - 8.2 事件处理的常用写法
 - 8.2.1 内部匿名类
 - 8.2.2 内部类
 - 8.2.3 实现接口的类
 - 8.2.4 继承的类的方法
 - 8.3 键盘事件
 - 8.3.1 键盘事件程序示例
 - 8.3.2 键盘事件的各种属性
 - 8.4 鼠标事件
 - 8.4.1 鼠标事件程序示例
 - 8.4.2 鼠标事件的各种属性
 - 8.5 其他常用的事件
 - 8.5.1 选中事件
 - 8.5.2 文本修改程序示例
 - 8.5.3 文本修改事件: VerifyEvent 的各种属性
 - 8.5.4 文本修改事件: VerifyEvent 和 ModifyEvent 的区别
 - 8.6 无类型的事件
 - 8.6.1 注册无类型事件监听器
 - 8.6.2 无类型事件程序示例
 - 8.7 本章小结
- 第3篇 SWT 高级篇
- 第9章 SWT 高级控件
 - 9.1 链接文本 (Link)
 - 9.2 菜单 (Menu 和 MenuItem)
 - 9.2.1 菜单与菜单项之间的关系
 - 9.2.2 菜单的样式
 - 9.2.3 菜单项的样式
 - 9.2.4 设置菜单项的图标
 - 9.2.5 设置菜单项快捷键

9.3	工具栏 (ToolBar和ToolItem)
9.3.1	工具栏图片资源的管理
9.3.2	工具栏的不同样式
9.3.3	工具栏按钮的不同样式
9.3.4	工具栏常用的方法
9.4	可拖动的工具栏 (CoolBar和CoolItem)
9.4.1	带有下拉选项的工具栏
9.4.2	常用的方法
9.5	系统托盘 (Tray和TrayItem)
9.6	滑动组件
9.6.1	滑块 (Slider)
9.6.2	刻度条 (Scale)
9.6.3	微调按钮 (Spinner)
9.7	进度条 (ProgressBar)
9.8	对话框
9.8.1	消息提示框 (MessageBox)
9.8.2	文件目录对话框 (DirectoryDialog)
9.8.3	文件对话框 (FileDialog)
9.8.4	颜色对话框 (ColorDialog)
9.8.5	字体对话框 (FontDialog)
9.8.6	打印对话框 (PrintDialog)
9.9	表格 (Table、TableItem和TableColumn)
9.9.1	Table、TableItem和TableColumn的关系
9.9.2	设置带有选择框的表格
9.9.3	设置可同时选中多行表格
9.9.4	可拖动的表格
9.9.5	设置单元格的图标
9.9.6	改变选中行高亮显示的颜色
9.9.7	带有上下文菜单的表格
9.9.8	可编辑的表格 (TableEditor)
9.9.9	用键盘控制表格 (TableCursor)
9.9.10	带有进度条的表格
9.9.11	表格小结
9.10	树 (Tree)
9.10.1	不同样式的树
9.10.2	为树添加图标
9.10.3	可编辑的树
9.10.4	表格树
9.10.5	树小结
9.11	格式化文本 (StyleText)
9.11.1	格式化对象 (StyleRange)
9.11.2	格式化文本的事件处理
9.11.3	对选中文本设置格式
9.11.4	自动为数字字符着色
9.11.5	换行自动设置背景颜色
9.12	浏览器
9.13	本章小结
第10章	SWT中的拖放支持
10.1	可拖放的树
10.2	拖放原理概述
10.3	拖放源 (DragSource)
10.3.1	创建拖放源对象
10.3.2	定义拖放源数据传输类型
10.3.3	处理拖放源事件
10.4	拖放目标 (DragTarget)
10.4.1	定义目标对象
10.4.2	定义目标对象的数据传输类型
10.4.3	处理拖放目标事件
10.5	传输数据 (Transfer)
10.6	综合示例：简单购物车
10.7	对剪贴板的操作
10.8	本章小结
第11章	SWT线程
11.1	线程概述
11.1.1	什么是线程
11.1.2	创建线程的两种方式
11.2	SWT中的UI线程
11.3	其他线程访问UI线程
11.4	改进的进度条
11.5	多线程程序设计
11.6	本章小结
第12章	SWT系统资源

- 12.1 系统资源概述
 - 12.1.1 什么是系统资源
 - 12.1.2 释放资源的原则
 - 12.1.3 访问资源的原则
 - 12.1.4 何时释放资源
 - 12.2 颜色 (Color)
 - 12.2.1 系统颜色
 - 12.2.2 RGB 颜色
 - 12.3 字体 (Font)
 - 12.4 光标 (Cursor)
 - 12.5 图像 (Image)
 - 12.5.1 画布类 (Canvas)
 - 12.5.2 图像类 (Image)
 - 12.5.3 图像数据类 (ImageData)
 - 12.5.4 保存图像类 (ImageLoader)
 - 12.5.5 Eclipse 的图标
 - 12.6 SWT 绘图
 - 12.6.1 使用绘制对象的方法
 - 12.6.2 绘制线条
 - 12.6.3 绘制字符
 - 12.6.4 绘制填充图形
 - 12.6.5 绘制图像
 - 12.7 本章小结
- 第13章 SWT 的高级应用
 - 13.1 打印支持
 - 13.1.1 打印类 (Printer) 和打印数据类 (PrinterData)
 - 13.1.2 打印程序示例概述
 - 13.1.3 打印程序示例：主窗口程序
 - 13.1.4 打印程序示例：打开文件程序
 - 13.1.5 打印程序示例：设置字体和颜色程序
 - 13.1.6 打印程序示例：打印文本的程序
 - 13.1.7 打印程序示例：打印文件后的效果预览
 - 13.2 使用应用程序
 - 13.3 对 AWT / Swing 程序的支持
 - 13.4 OLE 和 ActiveX 控件的支持
 - 13.4.1 OLE 控件的面板类 (OleFrame)
 - 13.4.2 OLE 控件类 (OleClientSite 和 OleControlSite)
 - 13.4.3 OLE 程序示例
 - 13.5 Pocket PC 应用
 - 13.6 Web 应用 SWT
 - 13.7 本章小结
- 第4篇 JFace 篇
- 第14章 JFace 概述
 - 14.1 配置 JFace 运行环境
 - 14.2 第一个 JFace 程序
 - 14.3 JFace 框架概述
 - 14.4 JFace 的包结构
 - 14.5 本章小结
- 第15章 应用程序窗口
 - 15.1 JFace 的窗口类 (Window 类)
 - 15.2 应用程序窗口 ApplicationWindow 类
 - 15.3 带有菜单栏的主程序窗口
 - 15.3.1 简单写字板程序示例
 - 15.3.2 添加菜单栏的基本步骤
 - 15.3.3 创建菜单项
 - 15.3.4 菜单项的事件处理
 - 15.4 带有工具栏的主程序窗口
 - 15.5 带有状态栏的主程序窗口
 - 15.6 其他处理事件的方法
 - 15.6.1 “新建”操作
 - 15.6.2 “保存”操作
 - 15.6.3 “另存为”操作
 - 15.6.4 “复制”、“剪切”和“粘贴”操作
 - 15.7 本章小结
- 第16章 JFace 对话框
 - 16.1 JFace 对话框概述
 - 16.2 信息提示对话框 (MessageDialog)
 - 16.2.1 创建信息提示对话框
 - 16.2.2 错误消息对话框
 - 16.2.3 确认消息对话框
 - 16.2.4 消息对话框
 - 16.2.5 询问对话框

- 16.2.6 警告对话框
 - 16.2.7 JFace的本地化
- 16.3 输入对话框 (InputDialog)
 - 16.3.1 创建输入对话框
 - 16.3.2 创建输入文本的验证类
 - 16.4 带有提示信息的对话框 (TitleAreaDialog)
 - 16.5 错误提示对话框 (ErrorDialog)
 - 16.5.1 创建错误提示对话框
 - 16.5.2 使用错误状态对象
 - 16.5.3 同时显示多个错误信息
 - 16.6 带有进度条的对话框 (ProgressMonitorDialog)
 - 16.7 自定义对话框
 - 16.7.1 自定义对话框程序示例
 - 16.7.2 自定义对话框的步骤
 - 16.8 本章小结
- 第17章 向导式对话框
 - 17.1 向导式对话框概述
 - 17.1.1 向导式对话框所涉及的类
 - 17.1.2 向导式对话框的常用方法
 - 17.2 简单的向导式对话框示例
 - 17.2.1 第一个问题向导页面
 - 17.2.2 第二个问题向导页面
 - 17.2.3 感谢向导页面
 - 17.2.4 创建向导
 - 17.2.5 创建测试程序
 - 17.3 保存对话框状态
 - 17.4 复杂的向导式对话框示例
 - 17.4.1 自定义向导页面
 - 17.4.2 为向导添加帮助
 - 17.5 向导式对话框的事件处理
 - 17.6 本章小结
- 第18章 首选项
 - 18.1 首选项概述
 - 18.2 保存首选项的设置
 - 18.2.1 首选项值的设置和获取
 - 18.2.2 保存首选项所涉及的事件
 - 18.3 显示首选项页面
 - 18.3.1 创建一个首选项页面
 - 18.3.2 创建首选项页面所对应的节点
 - 18.3.3 显示首选项对话框
 - 18.4 创建树型的导航菜单
 - 18.4.1 第一种方法
 - 18.4.2 第二种方法
 - 18.5 首选项的选项设置
 - 18.5.1 字段编辑器概述
 - 18.5.2 使用字段编辑器基本步骤
 - 18.5.3 布尔型字段编辑器 (BooleanFieldEditor)
 - 18.5.4 颜色字段编辑器 (ColorFieldEditor)
 - 18.5.5 字体字段编辑器 (FontFieldEditor)
 - 18.5.6 路径列表字段编辑器 (PathEditor)
 - 18.5.7 单选分组字段编辑器 (RadioGroupFieldEditor)
 - 18.5.8 刻度条字段编辑器 (ScaleFieldEditor)
 - 18.5.9 整数型字段编辑器 (IntegerFieldEditor)
 - 18.5.10 选择路径字段编辑器 (DirectoryFieldEditor)
 - 18.5.11 选择文件字段编辑器 (FileFieldEditor)
 - 18.6 自定义首选项页面
 - 18.7 首选项的事件处理
 - 18.8 本章小结
- 第19章 MVC的表格、树和列表
 - 19.1 MVC概述
 - 19.2 表格 (TableViewer)
 - 19.2.1 创建表格控制器 (Controller)
 - 19.2.2 创建表格模型 (Model)
 - 19.2.3 创建组织表格视图 (View)
 - 19.2.4 添加和删除数据
 - 19.2.5 增加表格排序功能
 - 19.2.6 增加表格过滤功能
 - 19.2.7 编辑表格单元
 - 19.2.8 表格的事件处理
 - 19.2.9 带有复选框表格 (CheckBoxTableViewer)
 - 19.3 树 (TreeViewer)
 - 19.3.1 树的基本性质

- 19.3.2 创建树 (TreeViewer)
- 19.3.3 对树的操作
- 19.4 树和表格的综合示例
 - 19.4.1 文件浏览器功能概述
 - 19.4.2 程序的整体框架
 - 19.4.3 初始化树
 - 19.4.4 初始化表格
 - 19.4.5 程序的事件处理
- 19.5 列表 ListView
- 19.6 本章小结
- 第20章 JFace的工具类
 - 20.1 JFace资源管理机制
 - 20.1.1 图像描述符 (ImageDescriptor)
 - 20.1.2 图像注册器 (ImageRegistry)
 - 20.1.3 字体描述符和字体注册器
 - 20.1.4 颜色描述符和颜色注册器
 - 20.1.5 JFace的资源管理器 (JFaceResources)
 - 20.1.6 字符转换工具类 (StringConverter)
 - 20.2 类型检查的工具类
 - 20.3 本章小结
- 第21章 文本处理
 - 21.1 文本处理概述
 - 21.2 项目实战: JavaScript 编辑器
 - 21.2.1 主窗口预览
 - 21.2.2 项目文件结构
 - 21.3 主窗口模块
 - 21.3.1 代码实现
 - 21.3.2 主窗口程序代码分析
 - 21.3.3 启动主窗口程序
 - 21.4 代码着色
 - 21.4.1 源代码配置类 (SourceViewerConfiguration)
 - 21.4.2 基于规则的代码扫描器类 (RuleBasedScanner)
 - 21.4.3 设置代码扫描规则
 - 21.4.4 提取类 (Token) 和文本属性类 (TextAttribute)
 - 21.5 内容辅助
 - 21.5.1 配置编辑器的内容助手
 - 21.5.2 内容辅助类
 - 21.5.3 辅助建议类 (CompletionProposal)
 - 21.6 文档的撤销与重复
 - 21.6.1 文档管理器对象 (DefaultUndoManager)
 - 21.6.2 撤销操作的实现
 - 21.6.3 恢复操作的实现
 - 21.7 查找与替换窗口
 - 21.7.1 窗口的界面设计
 - 21.7.2 查找功能的实现
 - 21.7.3 替换功能的实现
 - 21.8 首选项的对话框
 - 21.8.1 首选项页面的代码实现
 - 21.8.2 打开首选项页面的代码
 - 21.9 文件的打开、保存与打印
 - 21.9.1 打开文件
 - 21.9.2 保存文件
 - 21.9.3 打印文件
 - 21.10 帮助对话框
 - 21.11 其他的一些工具类
 - 21.11.1 事件管理类
 - 21.11.2 资源管理类
 - 21.11.3 程序中使用的常量
 - 21.12 本章小结
 - 第5篇 RCP应用篇
 - 第22章 富客户端平台 (RCP) 应用
 - 22.1 RCP概述
 - 22.1.1 什么是RCP
 - 22.1.2 RCP应用的现状
 - 22.2 第一个RCP项目
 - 22.2.1 创建插件项目
 - 22.2.2 运行RCP程序
 - 22.2.3 插件的文件清单
 - 22.2.4 MANIFEST.MF文件
 - 22.2.5 build.properties文件
 - 22.2.6 plugin.xml文件
 - 22.3 RCP运行的基本原理

- 22.3.1 插件类MyRCPPlugin
- 22.3.2 应用程序类Application
- 22.3.3 工作台窗口类
- 22.3.4 操作类
- 22.3.5 透视图类
- 22.4 创建扩展的基本方法
- 22.4.1 使用扩展向导创建
- 22.4.2 手工创建
- 22.4.3 获取扩展点的帮助
- 22.5 本章小结
- 第23章 RCP开发
- 23.1 扩展操作集(Action Set)
- 23.1.1 操作集扩展点
- 23.1.2 编写代码创建操作对象
- 23.1.3 编写代码创建操作的步骤
- 23.1.4 其他与操作有关的扩展点
- 23.2 扩展视图
- 23.2.1 视图扩展点
- 23.2.2 视图类
- 23.2.3 视图之间的交互
- 23.2.4 添加视图的工具栏
- 23.2.5 添加上下文菜单
- 23.3 扩展编辑器
- 23.3.1 编辑器扩展点
- 23.3.2 编辑器类
- 23.3.3 打开编辑器
- 23.3.4 添加编辑器的菜单和工具栏
- 23.3.5 多页编辑器
- 23.4 透视图
- 23.4.1 透视图的布局
- 23.4.2 透视图扩展点
- 23.4.3 透视图类
- 23.5 首选项
- 23.5.1 首选项扩展点
- 23.5.2 首选项页面扩展点
- 23.6 帮助文档
- 23.6.1 联机帮助文档扩展点
- 23.6.2 扩展配置
- 23.6.3 联机帮助的目录结构
- 23.6.4 添加动态帮助
- 23.7 RCP产品的发布
- 23.7.1 Eclipse产品配置
- 23.7.2 导出RCP产品
- 23.7.3 运行RCP产品
- 23.8 本章小结
- 第24章 Eclipse表单
- 24.1 Eclipse表单概述
- 24.1.1 什么是Eclipse表单
- 24.1.2 Eclipse表单的特性
- 24.1.3 Eclipse表单使用的类包
- 24.2 表单开发基础
- 24.2.1 视图中使用表单
- 24.2.2 多页编辑器中使用表单
- 24.2.3 SWT程序中使用表单
- 24.2.4 获得表单工具对象(FormToolkit)一般方法
- 24.3 表单的各种控件
- 24.3.1 可滚动的表单(ScrolledForm)
- 24.3.2 可折叠的面板(ExpandableComposite)
- 24.3.3 内容区(Section)
- 24.3.4 超链接(Hyperlink)
- 24.3.5 表单文本(FormText)
- 24.4 表单的布局管理器
- 24.4.1 表格布局(TableWrapLayout)
- 24.4.2 列布局(ColumnLayout)
- 24.5 表单的高级应用
- 24.5.1 Master/Details模式
- 24.5.2 实现Master/Detail示例程序
- 24.6 本章小结
- 第25章 项目实战——客户关系管理系统
- 25.1 系统概述
- 25.1.1 系统预览
- 25.1.2 基本概念介绍

- 2 5 . 1 . 3 系统的运行环境
- 2 5 . 1 . 4 系统文件结构的说明
- 2 5 . 2 U I 界面设计
- 2 5 . 3 业务层设计
 - 2 5 . 3 . 1 业务层服务的定义
 - 2 5 . 3 . 2 业务层的实现
 - 2 5 . 3 . 3 业务层服务的管理
 - 2 5 . 3 . 4 业务层U M L 图
 - 2 5 . 3 . 5 如何调用业务对象
- 2 5 . 4 数据库层设计
 - 2 5 . 4 . 1 数据库接口类
 - 2 5 . 4 . 2 实现了M y S Q L 数据库类
 - 2 5 . 4 . 3 如何调用数据访问对象
 - 2 5 . 4 . 4 应用多种数据库
 - 2 5 . 4 . 5 数据库的初始化的脚本
 - 2 5 . 4 . 6 表所对应的P O J O 类
- 2 5 . 5 登录模块
 - 2 5 . 5 . 1 系统的上下文对象保存登录状态
 - 2 5 . 5 . 2 登录验证的实现
 - 2 5 . 5 . 3 登录窗口的实现
- 2 5 . 6 主窗口界面
 - 2 5 . 6 . 1 工作台的实现
 - 2 5 . 6 . 2 系统托盘的实现
 - 2 5 . 6 . 3 菜单栏和工具栏的实现
 - 2 5 . 6 . 4 操作管理类(A c t i o n M a n a g e r)
 - 2 5 . 6 . 5 新建客户操作(N e w C u s t o m e r A c t i o n)
 - 2 5 . 6 . 6 打开视图操作(S h o w V i e w A c t i o n)
- 2 5 . 7 各种视图和编辑器的实现
 - 2 5 . 7 . 1 快速新建客户视图
 - 2 5 . 7 . 2 客户列表视图
 - 2 5 . 7 . 3 客户详细编辑器
 - 2 5 . 7 . 4 联系人列表视图
 - 2 5 . 7 . 5 快速新建联系人视图
 - 2 5 . 7 . 6 搜索视图
 - 2 5 . 7 . 7 导航视图
- 2 5 . 8 新建客户联系人向导
 - 2 5 . 8 . 1 新建客户向导
 - 2 5 . 8 . 2 新建联系人向导
- 2 5 . 9 首选项的实现
- 2 5 . 1 0 p l u g i n . x m l 文件清单
- 2 5 . 1 1 本章小结